SiCortex Ice9 Specification

This document is SiCortex Confidential.

May 14, 2014

Contents

1 Overview **41**
- 1.1 Some History . . . 41
- 1.2 The System . . . 42
- 1.3 ICE9 . . . 43
  - 1.3.1 Goals . . . 43
- 1.4 Overall Block Diagram . . . 44
  - 1.4.1 Processor Cores . . . 44
  - 1.4.2 L2 Cache . . . 44
  - 1.4.3 Memory Controller . . . 44
  - 1.4.4 PCI-Express Controller . . . 44
  - 1.4.5 Fabric . . . 44
    - 1.4.5.1 DMA Engine . . . 44
    - 1.4.5.2 Fabric Switch . . . 44
    - 1.4.5.3 Link Controllers . . . 46
    - 1.4.5.4 Link Subsystem . . . 46
  - 1.4.6 Clock Generator . . . 46
  - 1.4.7 Miscellaneous . . . 46
- 1.5 Latency Calculations . . . 46
  - 1.5.1 Links and Wire-Handling Latency . . . 46
  - 1.5.2 ICE9 to ICE9 Latency . . . 47
- 1.6 Address Map . . . 47

2 Internode Link **49**
- 2.1 Overview . . . 49
- 2.2 Differences, Bugs, and Enhancements . . . 50
  - 2.2.1 Product and Chip Pass Differences . . . 50
  - 2.2.2 Known Bugs and Possible Enhancements . . . 50
- 2.3 Reference Documents . . . 50
- 2.4 SERDES Fabric Links . . . 50
- 2.5 8B/10B code . . . 51
- 2.6 The Lane Transmitter (Txlane) . . . 51
  - 2.6.1 Synchronizer setup between sclk and txclkP . . . 52
  - 2.6.2 Txlane data latency estimates . . . 53
  - 2.6.3 Txlane module ports (This port list is not complete. Needs portlist Spec from AnalogBits) . 53
  - 2.6.4 8B10B code Validation Plan . . . 53
  - 2.6.5 Verification Checklist: (This section is not complete) . . . 53
- 2.7 The Lane Receiver (Rxlane) . . . 54
  - 2.7.1 Clock Alignment and Synchronizer setup between Rxlane and Framer transfer . . . 55
    - 2.7.1.1 SkipBeat Handshake . . . 55
    - 2.7.1.2 The RxClk alignment . . . 56
  - 2.7.2 The Framer Module . . . 56
    - 2.7.2.1 The clock alignment and synchronizer setup . . . 57
    - 2.7.2.2 Framing Function and flag-LaneHealth . . . 57
  - 2.7.3 The Wordsync function . . . 57
  - 2.7.4 Rxlane to Framer data latency estimates . . . 58

2.7.5 Rxlane module ports . . . 59
2.7.6 8B10B code Validation Plan . . . 59
2.7.7 Verification Checklist: . . . 59
2.8 The Fabric Link Receiver . . . 60
2.8.1 Status Flags required by RxLinkSync and RxLC . . . 61
2.8.2 RxLinkSync Routine . . . 62
2.8.3 Verification Checklist: . . . 63
2.9 The Fabric Link Transmitter . . . 63
2.9.1 Status Flags required by TxLC . . . 64
2.9.2 TxLinkSync Routine . . . 65
2.9.3 Verification Checklist: . . . 66
2.10 Reset bring-up sequence . . . 66
2.10.1 When do Link Registers Get Reset . . . 67
2.10.1.1 AnalogBits QPMA Registers . . . 67
2.10.1.2 QSC Registers . . . 67
2.10.1.3 FLT and FLR link Registers . . . 67
2.10.2 Enabling Links . . . 67
2.10.2.1 Determine QPMA Impedance Settings . . . 68
2.10.2.2 Configure QPMA Calibration Settings . . . 68
2.10.2.3 Initialize SkipBeat Functions . . . 68
2.10.2.4 Enable the Links . . . 68
2.11 Diagnostic Modes . . . 68
2.11.1 NearEndLoopback Mode . . . 69
2.11.1.1 Link-0 . . . 69
2.11.1.2 Link-1 . . . 69
2.11.1.3 Link-2 . . . 69
2.11.2 FarEndLoopback Mode . . . 69
2.11.3 Bit-Blasting Mode . . . 69
2.11.4 ATE Testing of Analogbits ABICDR43 . . . 70
2.11.5 PLL Bypass Mode Testing of Analogbits ABICDR43 . . . 70
2.12 Error recovery procedure . . . 71
2.12.1 Force Retraining . . . 71
2.13 Bring-Up Failure Points . . . 71
2.14 Registers That Can Prevent Link Coming Up . . . 74
2.15 Common Registers and Definitions . . . 75
2.15.1 Package Attributes . . . 75
2.15.2 Definitions . . . 75
2.15.3 Link Symbols . . . 75
2.15.4 Flr Events . . . 76
2.15.5 Flt Events . . . 76
2.16 FLT Registers . . . 76
2.16.1 R_FltxSoftReset . . . 76
2.16.2 R_Fltx FC Lane Control Register . . . 77
2.16.3 R_Fltx Lane Status . . . 78
2.16.4 R_FltxInvCFc . . . 78
2.16.5 R_FltxDispFc . . . 79
2.16.6 R_FltxAltNull . . . 79
2.16.7 R_FltxHeartbeat . . . 80
2.16.8 R_FltxDriveError . . . 80
2.16.9 R_FltxTxLcStatus . . . 81
2.16.10 R_FltxTxLcControl . . . 81
2.16.11 R_FltxTxLcCount . . . 81
2.16.12 R_FltxS2WaitTime . . . 82
2.16.13 Fltx Manual Override Rotator (MOR) . . . 82
2.16.14 R_FltxFarEndLoopback . . . 83
2.16.15 R_FltxBBDiag . . . 83

2.16.16 Fltx_BBDiagStatus . . . 84
2.17 FLR Registers . . . 85
2.17.1 R_FlrxSoftReset . . . 85
2.17.2 R_FlrxLinkStatus . . . 85
2.17.3 R_FlrxLinkControl . . . 86
2.17.4 R_FlrxRotator . . . 86
2.17.5 R_FlrxRxLcStatus . . . 86
2.17.6 R_FlrxLaneHealth . . . 87
2.17.7 R_FlrxWSyncMode . . . 87
2.17.8 R_FlrxWSyncStatus . . . 88
2.17.9 R_FlrxHeartbeat . . . 89
2.17.10 R_FlrxRxLcControl . . . 89
2.17.11 R_FlrxRxLcCount . . . 90
2.17.12 R_FlrxS2WaitTime . . . 90
2.17.13 Flrx Lane Invalid Character Error Register . . . 91
2.17.14 Flrx Lane Disparity Error Register . . . 91
2.17.15 R_Flrx Lane Status Register . . . 92
2.17.16 Flrx Lane Control Register . . . 92
2.17.17 Flrx Manual Override Rotator (MOR) . . . 93
2.17.18 R_FlrxBBDiag . . . 94
2.17.19 Flrx_BBDiagStatus . . . 95
2.18 FLR/FLT Register Allocation . . . 96
2.18.1 Flr0 . . . 96
2.18.2 Flr1 . . . 96
2.18.3 Flr2 . . . 96
2.18.4 Flt0 . . . 96
2.18.5 Flt1 . . . 97
2.18.6 Flt2 . . . 97
2.19 Quad Serdes Physical Media Access (QPMA) . . . 97
2.19.1 Calibration and Impedance Control of the driver and Receiver . . . 98
2.19.2 Verification Checklist: . . . 99
2.20 Quad Serdes Control (QSC) Registers . . . 99
2.20.1 R_QscGo . . . 99
2.20.2 R_QscStatus . . . 99
2.20.3 R_QscCA . . . 100
2.20.4 R_QscSerDatAR . . . 101
2.20.5 R_QscSerDatT . . . 101
2.20.6 R_QscSerDatP . . . 101
2.20.7 R_QscQpmaStatus . . . 101
2.20.8 R_QscQpmaImpCalibration . . . 102
2.20.9 R_QscQpmaControl . . . 103
2.20.10 R_QscQpmaTestControl . . . 104
2.20.11 R_QscInterrupt . . . 105
2.20.12 Qsc TxBBDiag . . . 106
2.20.13 Qsc Lane Status Register . . . 107
2.20.14 Qsc Lane Control Register . . . 108
2.20.15 R QscRxBBDiag . . . 109
2.20.16 R QscRxBBDiagStatus . . . 110
2.21 Link Unit Implementation Interface . . . 110
2.21.1 Interrupt Interface . . . 110
2.21.2 Serial Configuration Bus Interface . . . 110
2.21.3 Differential Drivers and Receivers . . . 110
2.21.4 Fabric Switch Interface . . . 110
2.21.5 The transmitter Handshake Ports . . . 111
2.21.6 The Receiver Handshake Ports . . . 113

3 The Dense Fabric Switch **115**
3.1 Overview . . . 115
3.1.1 Specifications . . . 115
3.2 Differences, Bugs, and Enhancements . . . 115
3.2.1 Product and Chip Pass Differences . . . 115
3.2.2 Known Bugs and Possible Enhancements . . . 115
3.3 Description . . . 116
3.3.1 Routing . . . 116
3.3.2 Virtual Channel Assignment . . . 116
3.3.3 Virtual Channel Arbitration . . . 117
3.3.4 Flow Control . . . 118
3.3.5 Error Control . . . 118
3.3.6 Out-of-Band Channel . . . 119
3.4 Operation . . . 119
3.4.1 The Data Link . . . 120
3.4.1.1 Fabric Packet Header Class . . . 120
3.4.1.2 Fabric Packet Trailer Class . . . 121
3.4.1.3 Fabric Data Packets . . . 121
3.4.1.4 Fabric Packet Idle Class . . . 121
3.4.2 The Control Link . . . 122
3.4.2.1 Fabric Control Packet Class . . . 122
3.4.3 Control Link Use . . . 122
3.4.4 Error Recovery . . . 123
3.4.5 Poison . . . 123
3.4.6 Mission Mode . . . 124
3.5 Special Communication Paths . . . 124
3.5.1 The Out-of-Band Communication Registers . . . 124
3.6 Deadlock Avoidance . . . 124
3.7 The Switch Architecture . . . 125
3.7.1 General Organization . . . 125
3.7.2 Ordering Requirements . . . 125
3.7.3 Local Arbitration: Within A Crosspoint Buffer . . . 125
3.7.4 Global Arbitration: Between Crosspoint Buffers . . . 126
3.7.5 Why Two Levels of Global Arbitration? . . . 128
3.7.6 Stitching it all Together . . . 128
3.8 Error Detection and Recovery . . . 128
3.8.1 CRC Generation and Checking . . . 129
3.8.2 Handling Poisoned Packets . . . 129
3.8.3 Transient Bit Errors on the Link . . . 129
3.8.4 Corrupted VC . . . 130
3.8.5 Corrupted Route . . . 130
3.8.6 Corrupted Buffer Index . . . 130
3.8.7 Corrupted LSN . . . 131
3.8.8 Misc. Bad Data (CRC Mismatch) . . . 131
3.8.9 Uncorrectable ECC Error in Packet Store or Replay Buffer . . . 131
3.8.10 Uncorrectable ECC Error on Data to DMA Engine . . . 131
3.8.11 Uncorrectable ECC Error on Data from DMA Engine . . . 131
3.8.12 Upstream Link Goes Down . . . 131
3.8.13 Downstream Link Goes Down . . . 131
3.9 The Control/Status Register Path . . . 132
3.10 Components and Hierarchy . . . 132
3.10.1 Switch Top level . . . 132
3.10.1.1 External Ports . . . 132
3.10.1.2 Serial Configuration Bus Interface . . . 133
3.10.1.3 Interrupt Outputs . . . 133
3.10.1.4 The DMA to Fabric Switch Interface . . . 134

3.10.1.5 The Fabric Link Receiver (FLR) to Switch Interface . . . 135
3.10.1.6 The Fabric Link Transmitter (FLT) to Switch Interface . . . 137
3.10.2 Interblock Signals . . . 138
3.10.3 The Input Block . . . 138
3.10.3.1 Error Detection and Recovery Table . . . 138
3.10.4 The Output Block . . . 140
3.10.5 The DMA Input Block . . . 140
3.10.5.1 Error Detection and Recovery Table . . . 140
3.10.6 The DMA Output Block . . . 141
3.10.7 The Crosspoint Buffer . . . 141
3.10.7.1 The Arbitration Array . . . 141
3.10.7.2 The Packet Store . . . 141
3.11 Pipeline Timing . . . 141
3.11.1 Summary . . . 142
3.11.2 Incoming Packet is Stored in Crosspoint Buffer, Arbitrates, and Wins . . . 142
3.11.3 Packet Must Wait for Available Downstream Buffer . . . 144
3.11.4 Packet Loses Global Arb, but Wins on Second Try . . . 144
3.11.5 Packet with CRC Error is Poisoned and Sent Anyway . . . 144
3.11.6 Packet with CRC Error is Dropped . . . 145
3.11.7 About the Bypass Paths . . . 146
3.11.8 3 Cycle Latency Path . . . 147
3.11.9 4 Cycle Latency Path . . . 147
3.11.10 5 Cycle Latency Path . . . 148
3.11.11 6 Cycle Latency Path (No Bypass) . . . 148
3.11.12 End of Control Packet Arrives, Packets are Acknowledged . . . 148
3.11.13 End of Control Packet Arrives with ErrFlag=1, Causing Replay . . . 149
3.12 FSW Registers and Definitions . . . 150
3.12.1 Package Attributes . . . 150
3.12.2 Definitions . . . 150
3.12.3 Output Mux Select Choices . . . 151
3.12.4 Replay State Machine . . . 151
3.12.5 Fabric Switch Control/Status Registers . . . 151
3.12.5.1 Block Reset Register . . . 152
3.12.5.2 Block Enable Register . . . 153
3.12.5.3 Input Block Mode Register . . . 153
3.12.5.4 Output Block Mode Register . . . 154
3.12.5.5 PoolMask Register . . . 154
3.12.5.6 Out-of-Band Upstream Register . . . 155
3.12.5.7 Out-of-Band Downstream Register . . . 155
3.12.5.8 Output Block Status Registers . . . 156
3.12.5.9 Force Error Register . . . 156
3.12.5.10 Bypass Enable Register . . . 157
3.12.5.11 Input Block Data Packet CRC Error Counter . . . 158
3.12.5.12 Input Block Idle Packet CRC Error Counter . . . 158
3.12.5.13 Input Block Good Packet Counter . . . 159
3.12.5.14 Input Block Poison Counter . . . 159
3.12.5.15 Output Block Control Packet Error Counter . . . 159
3.12.5.16 Output Block Replay Counter . . . 160
3.12.5.17 DMA Input Block Packet Counter . . . 160
3.12.5.18 DMA Output Block Packet Counter . . . 161
3.12.5.19 Upstream Control Packet Capture Registers . . . 161
3.12.5.20 Interrupt Cause Registers 0, 1, 2 . . . 162
3.12.5.21 Interrupt Cause Register 3 - For Crosspoint Buffer ECC Errors . . . 164
3.12.5.22 Interrupt Mask Registers . . . 164
3.12.5.23 Master Interrupt Register . . . 164
3.12.5.24 Model Magic Register . . . 165

3.13 Reset and Initialization . . . 165
3.14 Internal Data Formats and States . . . 165
3.14.1 Encoding of Buses between FswCsr and FswIbx . . . 165
3.14.1.1 CsrIbxStat - For csr_ibx_Stat_sa bus . . . 165
3.14.1.2 IbxCsrStat - For csr_ibx_Stat_sa bus . . . 166
3.14.2 SCB Performance Events . . . 167
3.14.3 Encoding of Buses between FswCsr and FswDmai . . . 168
3.14.3.1 CsrDmaiStat - For csr_dmai_Stat_sa bus . . . 168
3.14.3.2 DmaiCsrStat - For dmai_csr_Stat_sa bus . . . 169
3.14.4 Encoding of Buses between FswCsr and FswObx . . . 169
3.14.4.1 CsrObxStat - For csr_obx_Stat_sa bus . . . 169
3.14.4.2 ObxCsrStat - For obx_csr_Stat_sa bus . . . 170
3.14.5 Encoding of Buses between FswCsr and FswDmao . . . 171
3.14.5.1 CsrDmaoStat - For csr_dmao_Stat_sa bus . . . 171
3.14.5.2 DmaoCsrStat - For dmao_csr_Stat_sa bus . . . 172
3.14.6 Encoding of Buses between FswCsr and FswXbx . . . 172
3.14.6.1 CsrXbxStat - For csr_xbx_Stat_sa bus . . . 172
3.14.6.2 XbxCsrStat - For xbx_csr_Stat_sa bus . . . 172
3.14.7 Open issues . . . 173

4 DMA Engine Microcode 175
4.0.8 Package Attributes . . . 175
4.1 Introduction . . . 175
4.2 Goals . . . 176
4.3 Differences, Bugs, and Enhancements . . . 176
4.3.1 Product and Chip Pass Differences . . . 176
4.3.2 Known Bugs and Possible Enhancements . . . 177
4.4 Model . . . 177
4.4.1 Terminology . . . 177
4.4.1.1 DMA Context (formerly Process) . . . 177
4.4.1.2 Thread . . . 177
4.4.1.3 Handle . . . 177
4.4.1.4 Packet . . . 177
4.4.1.5 Command . . . 178
4.4.1.6 Segment . . . 178
4.4.1.7 Errors . . . 178
4.4.1.8 Transmit . . . 178
4.4.1.9 Receive . . . 178
4.4.1.10 Multicast . . . 179
4.4.1.11 Collective . . . 179
4.4.1.12 Copy . . . 179
4.4.2 High-level Hardware View . . . 179
4.4.3 Canonical MPI Transfer Patterns . . . 179
4.4.3.1 Eager Transfer . . . 179
4.4.3.2 Single-ended Messages . . . 180
4.4.3.3 Rendezvous Exchange . . . 180
4.5 Queues . . . 182
4.5.1 Command and Port queues . . . 182
4.5.1.1 Process quota . . . 184
4.5.1.2 Command order . . . 184
4.5.2 Event queue . . . 184
4.5.2.1 Hardware-generated events . . . 184
4.5.3 Summary of DMA Engine Queues . . . 185
4.6 Modes of Operation . . . 186
4.6.1 Synchronous mode . . . 186
4.6.2 Asynchronous mode . . . 186

4.6.3 Interrupt mode . . . 187
4.6.4 Fabric Processor . . . 187
4.6.5 Virtualized mode . . . 187
4.7 Communication state . . . 187
4.7.1 Transmit state . . . 187
4.7.2 Receive state . . . 188
4.7.3 Notifiers . . . 188
4.7.4 Buffer descriptor . . . 188
4.7.4.1 Virtual Memory swapping . . . 189
4.7.5 Route descriptor . . . 189
4.7.6 Heap . . . 190
4.7.7 Protected data structures . . . 191
4.7.8 DMA Engine Common Control/Status . . . 192
4.8 Commands . . . 195
4.8.1 Command Header . . . 195
4.8.2 Send_Event Command . . . 195
4.8.3 Send_Cmd Command . . . 196
4.8.4 Do_Cmd Command . . . 196
4.8.5 Put_Bf_Bf Command . . . 197
4.8.6 Put_Im_Hp Command . . . 198
4.8.7 Supervise Command . . . 199
4.8.8 Undefined Commands . . . 199
4.9 Packet formats . . . 199
4.9.1 Packet header and check . . . 199
4.9.2 Packet Types . . . 200
4.9.3 Direct Transmission: Enq_Direct . . . 200
4.9.4 DMA . . . 200
4.9.5 DMA_End . . . 201
4.9.6 Wr_Heap . . . 201
4.9.7 Enq_Response . . . 202
4.9.8 Poison . . . 202
4.10 Notes on Complex Functions . . . 203
4.10.1 Rendezvous . . . 203
4.10.2 Stride and Scatter/Gather . . . 203
4.10.3 Barrier and Collective . . . 203
4.10.4 Multicast . . . 203
4.10.5 Out-of-band . . . 204
4.10.6 Receive Matching . . . 204
4.10.7 Initialization . . . 204
4.10.7.1 Black Hole . . . 204
4.10.7.2 Reset . . . 204
4.10.7.3 Microcode load . . . 204
4.10.7.4 Variable binding . . . 204
4.10.7.5 Initialization of common resources . . . 205
4.10.7.6 Initialization of process resources . . . 205
4.10.8 Process Rundown . . . 206
4.11 Lessons for Next Time . . . 206
4.11.1 Queue Manager . . . 206
4.11.2 Additional functionality . . . 206
4.11.2.1 Enqueue/Dequeue commands . . . 206
4.11.2.2 Global locks . . . 206
4.11.3 Microcode . . . 206
4.11.3.1 Buffer addressing . . . 206
4.11.3.2 Buffer reset . . . 206
4.11.4 Copy port . . . 207
4.11.5 Receive ports . . . 207

4.11.6 Cache 207
4.12 Microcode 207

5 DMA Engine 209
5.0.1 Package Attributes 209
5.1 Introduction 209
5.2 Implementation 209
5.2.1 Top Level Block Diagram 210
5.2.2 External Interfaces 211
5.2.2.1 Fabric Switch to DMA receive port X (X=0,1,2) 211
5.2.2.2 DMA transmit port X to Fabric Switch (X=0,1,2) 211
5.2.2.3 DMA to L2 Cache Switch 211
5.2.3 Module Hierarchy 211
5.2.4 DmaUe: Microengine Control Logic 212
5.2.5 DmaImem: Microengine Instruction Memory 213
5.2.6 DmaAlu: Microengine ALU 214
5.2.7 DmaDmem: Microengine Data Memory 214
5.2.8 DmaRxp: Receive Ports 218
5.2.9 DmaTxp: Transmit Ports 220
5.2.10 DmaCopy: Copy Port 221
5.2.11 DmaCif: Cache Interface 222
5.2.11.1 Cache Interface Queues 225
5.2.11.2 Interfaces in DmaCif 226
5.2.11.3 TaskStart Interface (Microengine to DmaCif) 226
5.2.11.4 StartIo Interface (DmaCif to microengine) 226
5.2.11.5 Interface to L2 Cache 229
5.2.11.6 Cycle Behavior: TaskStart to CmdAddr Bus 229
5.2.11.7 Memory to DMA Pipeline 231
5.2.11.8 I/O Access Pipeline (Read and Write) 231
5.2.11.9 I/O Write Pipeline 232
5.2.11.10 Task interface pipeline 232
5.2.12 Microengine Programming 232
5.2.12.1 Instructions 232
5.2.12.2 Operand selection 233
5.2.12.3 Destination Selection 233
5.2.12.4 ALU operations 233
5.2.12.5 Sleep Functions 234
5.2.12.6 Stall 235
5.2.12.7 Memory Transfer 235
5.2.12.8 Branch Functions 235
5.2.12.9 Next Address 236
5.2.13 Unified Engine 236
5.2.14 Bandwidth 236
5.2.15 Matching 236
5.2.16 Interface registers 237
5.2.17 Coherence 237
5.2.18 Alignment 237
5.2.19 Strides and Scatter/Gather 237
5.2.20 Output Thread 237
5.2.21 Input Thread 238
5.2.22 Thread performance 238
5.2.23 Queue manager 238
5.2.24 Port manager 239
5.2.25 Copy Thread 239
5.2.26 Timeouts 239
5.2.27 Error Conditions 239

5.3 Notes . . . 240
5.3.1 Rendezvous . . . 240
5.3.2 Ethernet simulation . . . 240
5.3.3 Barrier . . . 240
5.3.4 Cache interface . . . 241
5.3.5 Performance Counters . . . 241
5.4 Registers and Definitions . . . 242
5.5 Microengine Instructions . . . 242
5.5.1 Instruction Fields . . . 242
5.5.2 Operand A addressing modes . . . 242
5.5.3 Operand B addressing modes . . . 243
5.5.4 Destination Addressing Modes . . . 243
5.5.5 Special Registers addressed by Operand A . . . 244
5.5.6 Special Registers addressed by Operand B . . . 244
5.5.7 Special Registers addressed by Destination . . . 245
5.5.8 ALU Operation Field . . . 247
5.5.9 Memory Operation Field . . . 249
5.5.10 Memory Transfer Length Selection . . . 249
5.5.11 Sleep Mode Field . . . 249
5.5.12 Sleep Index Field, when Sleep=HwFlag . . . 250
5.5.13 Sleep Index Field, when Sleep=TakeMutex or DropMutex . . . 250
5.5.14 Internal Encoding of Sleep Conditions . . . 251
5.5.15 Branch Field . . . 252
5.5.16 Dedicated Microinstruction Addresses . . . 253
5.5.17 Miscellaenous Constant Definitions . . . 253
5.5.18 DMA Thread Numbers . . . 255
5.5.19 DMA Port numbers . . . 255
5.5.20 DMA Queue numbers . . . 255
5.5.21 DMA Internal Memory Addresses . . . 255
5.5.22 DMA Internal Memory Addresses (Mem Field) . . . 256
5.5.23 Receive Port Buffer State Machine . . . 256
5.5.24 Receive Port CMUX Select Values . . . 257
5.5.25 Transmit Port Buffer State Machine . . . 257
5.5.26 Transmit Port: Packet Builder State Machine . . . 257
5.5.27 Copy Port Buffer State Machine . . . 257
5.5.28 Copy Port: Read/Write Memory Buffer Address . . . 257
5.5.29 Dma Cache Interface Task . . . 258
5.5.30 Dma Cache Interface: Memory Operation Type . . . 258
5.5.31 Dma Cache Interface: Type of Task . . . 259
5.5.32 Dma Cache Interface: Numbering of Queues . . . 260
5.5.33 Dma Cache Interface: Depth of Queues for ICE9 . . . 260
5.5.34 Dma Cache Interface: Depth of Queues for TWC9 . . . 261
5.5.35 Dma Cache Interface: Outstanding Read Table entry . . . 261
5.5.36 Dma Cache Interface: Outstanding Write Table entry . . . 261
5.5.37 Dma Cache Interface: Block Read Retry Queue (BrdrQ) for ICE9 . . . 261
5.5.38 Dma Cache Interface: Block Read Retry Queue (BrdrQ) for TWC9 . . . 262
5.5.39 Dma Cache Interface: Command RDIO Queue (CrdioQ) . . . 262
5.5.40 Dma Cache Interface: SPCL/INT Queue (CSpclIntQ) for ICE9 . . . 262
5.5.41 Dma Cache Interface: SPCL/INT Queue (CSpclIntQ) for TWC9 . . . 262
5.5.42 Dma Cache Interface: Data Response Queue (DataRspQ) . . . 262
5.5.43 Dma Cache Interface: Data Write Queue (DWQ) . . . 262
5.5.44 Dma Cache Interface: I/O Read Queue (DRDIOQ) . . . 262
5.5.45 Dma Cache Interface: StartIoQ for ICE9 . . . 263
5.5.46 Dma Cache Interface: StartIoQ for TWC9 . . . 263
5.5.47 Dma Cache Interface: StartIoType . . . 263
5.5.48 Dma Cache Interface: Address memory entry . . . 263

5.5.49 Dma Cache Interface: MemOut Address Sequencer States . . . 263
5.5.50 Dma Cache Interface: MemIn Address Sequencer States . . . 264
5.5.51 Internal Encodings for Microengine Operands . . . 264
5.5.52 I/O Region Type (DmaIoRegionType) . . . 264
5.5.53 External I/O Addresses . . . 264
5.6 Registers Accessible by RDIO/WTIO from Processors . . . 265
5.6.1 DMA Instruction Memory (IMEM) . . . 265
5.6.2 DMA Data Memory (DMEM) . . . 265
5.6.3 DMA Thread Select Register . . . 265
5.6.4 DMA Thread Pointer Registers . . . 266
5.6.5 DMA Thread Program Counter Registers . . . 266
5.6.6 DMA Programmable I/O Control Register . . . 267
5.6.7 DMA Application Interface Region 0 . . . 267
5.6.8 DMA Application Interface Region 1 . . . 267
5.7 Registers Accessible by Serial Configuration Bus . . . 268
5.7.0.1 Block Reset Register . . . 268
5.7.0.2 ECC Mode Register . . . 268
5.7.0.3 ALU Merge Operation Control Registers (added in Twice9) . . . 269
5.7.0.4 Force Error Register . . . 269
5.7.0.5 Microengine Status Registers . . . 270
5.7.0.6 Cache Interface Status Registers . . . 271
5.7.0.7 Rx/Tx Port Status Registers . . . 271
5.7.0.8 Copy Port Status Register . . . 272
5.7.0.9 Interrupt Cause Register . . . 272
5.7.0.10 Interrupt Mask Register . . . 274
5.8 SCB Performance Events . . . 274
5.9 Internal Data Formats and States . . . 275
5.9.1 Encoding of Buses between DmaCsr and DmaUe . . . 275
5.9.1.1 CsrUeStat - For csr_ue_Stat_ca bus . . . 275
5.9.1.2 UeCsrStat - For csr_ue_Stat_ca bus . . . 275
5.9.2 Encoding of Buses between DmaCsr and DmaCif . . . 276
5.9.2.1 CsrCifStat - For csr_cif_Stat_ca bus . . . 276
5.9.2.2 CifCsrStat - For csr_cif_Stat_ca bus . . . 276
5.9.3 Encoding of Buses between DmaCsr and DmaDmem . . . 276
5.9.3.1 CsrDmemStat - For csr_dmem_Stat_ca bus . . . 276
5.9.3.2 DmemCsrStat - For csr_dmem_Stat_ca bus . . . 277
5.9.4 Encoding of Buses between DmaCsr and DmaTxp . . . 277
5.9.4.1 CsrTxpStat - For csr_txp_Stat_ca bus . . . 277
5.9.4.2 TxpCsrStat - For csr_txp_Stat_ca bus . . . 277
5.9.5 Encoding of Buses between DmaCsr and DmaRxp . . . 277
5.9.5.1 CsrRxpStat - For csr_rxp_Stat_ca bus . . . 277
5.9.5.2 RxpCsrStat - For csr_rxp_Stat_ca bus . . . 277
5.9.6 Encoding of Buses between DmaCsr and DmaCopy . . . 278
5.9.6.1 CsrCopyStat - For csr_copy_Stat_ca bus . . . 278
5.9.6.2 CopyCsrStat - For csr_copy_Stat_ca bus . . . 278

6 Processor Segments 279
6.1 Overview . . . 279
6.2 Specifications . . . 279
6.3 User Code Visiable Bugs and Enhancements . . . 280
6.3.1 Product and Chip Pass Differences . . . 280
6.3.2 Known Bugs and Possible Enhancements (M5KF only) . . . 281
6.4 Kernel and Performance Bugs and Enhancements . . . 281
6.4.1 Product and Chip Pass Differences . . . 281
6.4.2 Known Bugs and Possible Enhancements (M5KF only) . . . 281
6.5 Complete Documentation . . . 282

6.6 BIU Description . . . 282
6.6.1 BIU Ports . . . 282
6.6.2 D-Cache Reads . . . 284
6.6.3 I-Cache Reads . . . 284
6.6.4 Istream Initial Reads . . . 285
6.6.5 Evictions . . . 285
6.6.6 IO Writes . . . 285
6.6.6.1 IO Write Buffer Counter . . . 285
6.6.7 Cache Instructions . . . 286
6.6.8 Prefetch Instruction . . . 286
6.6.9 Sync Instruction . . . 286
6.6.10 Load Linked and Store Conditional . . . 286
6.7 Interventions . . . 287
6.7.1 Intervention Deadlock Avoidance . . . 288
6.7.2 Example Intervention Cases . . . 288
6.8 WAIT . . . 288
6.9 Interrupts . . . 289
6.10 EJTag . . . 289
6.11 D Cache ECC . . . 289
6.12 Scheduling Hazards . . . 289
6.13 Dual Issue . . . 289
6.14 Floating Point Pipeline Enhancements . . . 290
6.14.1 Floating Point Repeat Rate and Latency . . . 291
6.15 The L2 Cache Segment and Pipelines . . . 291
6.15.1 The Tag Lookup . . . 291
6.15.2 The L2 Miss Data Pipeline . . . 295
6.15.3 L1 Updates Writebacks and Misses . . . 295
6.15.4 CSW Probe Operations . . . 297
6.15.5 Putting It All Together . . . 300
6.15.6 The SLC (slick) and Processor Access Stalls . . . 300
6.16 Initial Program Load and Processor Start-up . . . 300
6.17 Memory and IO Ordering Rules and Behavior . . . 300
6.18 I/O Accesses and Address Decoding . . . 303
6.18.1 CAC Local IO Registers . . . 303
6.18.2 CAC Remotely Accessible IO Registers . . . 303
6.19 Interrupts, Again . . . 303
6.19.1 CPU Interrupt lines . . . 303
6.19.2 The Interrupt Cause Registers . . . 303
6.19.3 The CSW INT Transaction and Writing the Interrupt Cause Registers . . . 304
6.19.4 Interprocessor Interrupts . . . 304
6.19.5 Machine Check Interrupts . . . 304
6.19.6 "Slow" Interrupts . . . 304
6.19.7 Delivering Interrupts to Other Processors . . . 304
6.20 Error Correction, Detection, Control, and Testing . . . 304
6.21 Processor/L2 Transactions – NittyGritty Details . . . 305
6.21.1 Processor L1 Cache Read Miss . . . 305
6.21.1.1 I-Stream Read L1 Miss, L2 Hit . . . 305
6.21.1.2 I-Stream Read L1 Miss, L2 Miss . . . 305
6.21.1.3 D-Stream Read L1 Miss, L2 Hit . . . 305
6.21.1.4 D-Stream Read L1 Miss, L2 Miss . . . 306
6.21.2 Processor L1 Cache Write Miss . . . 306
6.21.3 Processor L1 Cache Bypass Read to Cacheable Memory . . . 306
6.21.4 Processor L1 Cache Bypass Write to Cacheable Memory . . . 306
6.21.5 Processor I/O Read . . . 306
6.21.6 Processor I/O Write . . . 306
6.21.7 Processor L1 Eviction . . . 307

6.21.8 L2 Probe to Processor . . . 307
6.21.8.1 Probe Hits on Clean Block . . . 307
6.21.8.2 Probe Hits on Dirty Block . . . 307
6.21.8.3 Probe Misses in L1 . . . 307
6.22 L2 Responses to Probe Requests . . . 308
6.22.1 PRBINV . . . 308
6.22.2 PRBWIN . . . 308
6.22.3 PRBBRD . . . 308
6.22.4 PRBBWT . . . 314
6.22.5 PRBSHR . . . 314
6.23 L2 Responses to Other CSW Commands . . . 314
6.23.1 PRBNOHIT . . . 314
6.23.2 RDIO . . . 314
6.23.3 WTIO . . . 314
6.23.4 INT . . . 317
6.23.5 Incoming Data Completing a Memory Read Operation . . . 317
6.24 Registers and Definitions . . . 317
6.24.1 Package Attributes . . . 317
6.24.1.1 Package . . . 317
6.24.2 Definitions . . . 317
6.24.3 Register List . . . 318
6.24.4 Prefetch Hint Encodings . . . 319
6.24.5 CPU Performance Counter Events . . . 319
6.24.6 SCB Performance Core Events . . . 322
6.24.7 SCB Performance Events . . . 323
6.24.8 CpuConfig Register . . . 326
6.24.9 CpuConfig1 Register . . . 327
6.24.10 CpuConfig2 Register . . . 327
6.24.11 CpuFCCR Register . . . 328
6.24.12 CpuWatchLo Register . . . 328
6.24.13 CpuWatchHi Register . . . 328
6.24.14 CpuFEXR Register . . . 328
6.24.15 CpuXContext Register . . . 328
6.24.16 CpuDebug Register . . . 328
6.24.17 CpuDEPC Register . . . 329
6.24.18 CpuPerfCnt Register . . . 329
6.24.19 CpuPerfVPC Register . . . 330
6.24.20 CpuPerfPEA Register . . . 330
6.24.21 CpuFENR Register . . . 330
6.24.22 CpuErrCtl Register . . . 330
6.24.23 CpuCacheErr Register . . . 331
6.24.24 CpuTagLo Register . . . 331
6.24.25 CpuDataLo Register . . . 331
6.24.26 CpuDataHi Register . . . 332
6.24.27 CpuErrorEPC Register . . . 332
6.24.28 CpuDESAVE Register . . . 332
6.24.29 CpuDCR Register . . . 332
6.24.30 CpuFCSR Register . . . 332
6.24.31 CpuIBS Register . . . 333
6.24.32 CpuIBA Register . . . 333
6.24.33 CpuIBM Register . . . 333
6.24.34 CpuIBASID Register . . . 333
6.24.35 CpuIBC Register . . . 333
6.24.36 CpuDBS Register . . . 334
6.24.37 CpuDBA Register . . . 334
6.24.38 CpuDBM Register . . . 334

6.24.39 CpuDBASEID Register . . . 334
6.24.40 CpuDBC Register . . . 334
6.24.41 CpuDBV Register . . . 334
6.24.42 CpuIndex Register . . . 335
6.24.43 CpuRandom Register . . . 335
6.24.44 CpuEntryLo Register . . . 335
6.24.45 CpuContext Register . . . 335
6.24.46 CpuPageMask Register . . . 335
6.24.47 CpuWired Register . . . 335
6.24.48 CpuBadVAddr Register . . . 336
6.24.49 CpuFIR Register . . . 336
6.24.50 CpuCount Register . . . 336
6.24.51 CpuEntryHi Register . . . 336
6.24.52 CpuCompare Register . . . 336
6.24.53 CpuStatus Register . . . 336
6.24.54 CpuCause Register . . . 337
6.24.55 CpuEPC Register . . . 337
6.24.56 CpuPRId Register . . . 337
6.24.57 Ecc Injection Magic Register . . . 338
6.25 EJTAG Registers and Definitions . . . 338
6.25.1 EJTAG TAP Instructions . . . 338
6.25.2 CpuTapIDCODE Register . . . 339
6.25.3 CpuTapIMPCODE Register . . . 339
6.25.4 CpuTapDATA Register . . . 339
6.25.5 CpuTapADDRESS Register . . . 340
6.25.6 CpuTapECR Register . . . 340
6.25.7 CpuTapFASTDATA Register . . . 340
6.26 Cpu Implementation-Only Definitions . . . 340
6.26.1 Request Commands . . . 340
6.27 Cac Registers and Definitions . . . 341
6.27.1 Probe Queue Handler States . . . 341
6.27.2 Processor Interface Ready State Machine . . . 341
6.27.3 L2 Cache Pause During Fill State Machine . . . 342

7 L2 Cache Coherence and Switch 343
7.1 Summary . . . 343
7.2 Differences, Bugs, and Enhancements . . . 343
7.2.1 Product and Chip Pass Differences . . . 343
7.2.2 Known Bugs and Possible Enhancements . . . 344
7.3 L2 Cache Features . . . 344
7.3.1 Terminology . . . 344
7.3.2 Unusual Features . . . 345
7.3.3 Error Control . . . 346
7.4 Processor to L2 Cache Interface . . . 346
7.5 Major Blocks and the General Approach . . . 346
7.5.1 Supported Operations . . . 346
7.5.2 Per-Processor Segment . . . 346
7.5.3 Bidirectional spine structure . . . 347
7.5.4 Tags . . . 348
7.5.5 Hashed Index . . . 348
7.5.6 Outstanding Read CAM (ORC) and Write Back CAM (WBC) . . . 348
7.5.7 Victim Buffer . . . 348
7.6 I/O and DMA Transactions . . . 348
7.7 Coherence Interactions . . . 349
7.7.1 Races . . . 349
7.7.2 Probes . . . 349

7.8 Multiprocessor Issues . . . 349
7.8.1 LL/SC . . . 349
7.8.2 Lockstep cache thrashing . . . 349
7.8.3 Deadlock Freedom . . . 350
7.9 L2 Segment to Memory Interface . . . 350
7.9.1 Transaction ID . . . 350
7.9.2 Target . . . 353
7.9.3 Completion . . . 353
7.9.4 CSW Bus Arbitration . . . 353
7.9.4.1 Fairness . . . 354
7.9.4.2 Worst Case Traffic Analysis . . . 354
7.9.5 CSW Queuing of Commands and Data . . . 354
7.9.6 Transfer order . . . 354
7.10 Detailed Interface and Block Descriptions . . . 357
7.10.1 The Normal Flow Of Events, Hazards, and General Ordering Cases . . . 357
7.10.2 Transaction Steps and the CSW Buses . . . 358
7.10.3 The Outstanding Read CAM and the Write Back CAM . . . 363
7.10.3.1 The ORC . . . 364
7.10.3.2 The WBC . . . 364
7.10.4 Transaction Flows . . . 364
7.10.4.1 D-Stream Read to a Non Resident Block . . . 364
7.10.4.2 D-stream Read to a Cached Block . . . 369
7.10.4.3 I-stream Read to a Non Resident Block . . . 375
7.10.4.4 I-stream Read to a Cached Block . . . 379
7.10.4.5 D-stream Read to a Cached Block in SHARED State . . . 389
7.10.4.6 D-Stream Write Miss . . . 393
7.10.4.7 D-Stream Write to Invalidate . . . 393
7.10.4.8 Block Write to a Non Resident Block . . . 396
7.10.4.9 Block Write to a Cached Block . . . 398
7.10.4.10 Block Write to SHARED Location . . . 404
7.10.4.11 Block Write and Other Probe Collisions with Victimization . . . 406
7.10.4.12 Block Read to a Non Resident Block . . . 407
7.10.4.13 Block Read to a Cached Block . . . 410
7.10.4.14 Read from an I/O Location . . . 415
7.10.4.15 Write to an I/O Location . . . 416
7.10.4.16 Read after Read Hazard . . . 417
7.10.4.17 Read after Write Hazard . . . 420
7.10.4.18 Write After Read Hazards . . . 421
7.10.4.19 Write After Write Hazards . . . 421
7.10.5 Interrupt Delivery . . . 422
7.10.6 Special Communication Commands . . . 422
7.10.7 WINV, Victim Writebacks and the WriteBack CAM . . . 424
7.11 WRSTRANS and When Bad Things Happen to Good Blocks . . . 424
7.12 One Thousand Ships, One Thousand Nights . . . 425
7.12.1 Read Retry vs. Victim Writebacks . . . 425
7.12.2 PRBWIN A followed by RDEX A . . . 425
7.12.3 PRBXXX A While A Is Being Evicted . . . 426
7.12.3.1 PRBWIN Against an Evicted Block . . . 426
7.12.3.2 PRBSHR Against an Evicted Block . . . 426
7.12.3.3 PRBBWT Against an Evicted Block . . . 426
7.12.3.4 PRBBRD Against an Evicted Block . . . 427
7.12.3.5 PRBINV Against an Evicted Block . . . 427
7.12.4 PRBXXX A Just Prior to Evict Attempt on A . . . 427
7.12.5 Implications for Stimulus Generators and Checkers . . . 428
7.12.5.1 NOHIT sequencing against writeback data . . . 428
7.13 Command Fields . . . 428

7.14 Transaction IDs (TIDs) and TID Busy Signals . . . 428
7.14.1 TID Allocation – the IO and MEM TID Spaces . . . 429
7.15 The Parts . . . 430
7.15.1 The Coherence Controller (COH) . . . 430
7.15.1.1 Block Diagram . . . 430
7.15.1.2 Processing Pipeline(s) . . . 431
7.15.1.3 Recovering from Tag ECC Errors . . . 432
7.15.2 The L2 Switch (CSW) . . . 432
7.15.2.1 Bus Stops, Node Numbers, and Transaction Targets . . . 432
7.16 Arbitration at the PS to CSW Port . . . 432
7.17 Definitions and Enumerations . . . 437
7.17.1 Package Attributes . . . 437
7.17.2 Definitions . . . 437
7.17.3 Processor to L2 Cache Commands . . . 438
7.17.4 L2 Cache to Processor Commands . . . 438
7.17.5 L2 Cache to/from Coherence Controller Commands . . . 438
7.17.6 L2 Cache Coherence Widget States . . . 438
7.17.7 L2 Segment Cache States . . . 439
7.17.8 L2 Cache Modified States . . . 439
7.17.9 L2 Half Block Update Tags . . . 439
7.17.10 L2 Cache Interface Numbers (Bus Stop Numbers) . . . 439
7.17.11 L2 Cache Interface Numbers (Bus Stop Numbers) for TWICE9 . . . 439
7.17.12 Transaction IDs . . . 440
7.17.13 Transaction IDs for TWICE9 . . . 441
7.17.14 Address Tag and Index Fields for L2 and Coh Tag and Data arrays . . . 442
7.17.15 L2 Cache Useful Dimensions . . . 443
7.17.16 Coherence Engine Useful Dimensions . . . 443
7.17.17 Coherence Engine Useful Dimensions for Twice9A . . . 443
7.17.18 Coherence Engine L2 Tag Array Fields . . . 443
7.17.19 SPCL Address Request Fields . . . 444
7.17.20 SPCL CSW Command Fields . . . 444
7.18 Registers . . . 444
7.18.1 Cache Probe Control Register . . . 444
7.18.2 Cache Probe Address Register . . . 445
7.18.3 Cache Probe Random Address Registers . . . 445
7.18.4 Cache ECC Injection Register . . . 446
7.18.5 I/O Addresses in L2 Segment . . . 446
7.18.6 Interrupt Cause Register . . . 446
7.18.7 Interrupt Delivery Register . . . 447
7.18.8 Slow Interrupt Selection Register . . . 447
7.18.9 Slow Interrupt Status Register . . . 448
7.18.10 L2 Cache ECC Mode Register . . . 449
7.18.11 L2 Cache ECC Test Register . . . 449
7.18.12 L2 Cache Status Register . . . 450
7.18.13 L2 Cache Data ECC Error Address Register . . . 450
7.18.14 CSW ECC Error Address Register . . . 451
7.18.15 L2 Cache Tag ECC Error Address Register . . . 451
7.18.16 L2 Cache ECC Error Syndrome Register . . . 451
7.18.17 L2 Cache Send SPCL Request Address Range . . . 452
7.18.18 Coherence Engine ECC Mode Register . . . 452
7.18.19 Coherence Engine ECC Test Register . . . 453
7.18.20 Coherence Engine ECC Status Register . . . 453
7.18.21 Coherence Engine ECC Error Address Register . . . 453
7.18.22 Twice9+ Coherence Engine ECC Error Address Register . . . 454
7.18.23 Coherence Engine ECC Error Syndrome Register . . . 454
7.18.24 Coherence Engine Active Processor Segment Register . . . 454

7.19 Register Allocation . . . 455
7.19.1 CacLoc . . . 455
7.19.2 Coho . . . 455
7.19.3 Cohe . . . 455

8 Memory Controller 457
8.1 Overview . . . 457
8.2 Differences, Bugs, and Enhancements . . . 457
8.2.1 Product and Chip Pass Differences . . . 457
8.2.2 Known Bugs and Possible Enhancements . . . 458
8.3 General Description . . . 458
8.3.1 Clocks . . . 458
8.3.2 Reset and Initialization . . . 458
8.3.3 Serial Presence Detect . . . 459
8.3.4 PHY Read Path DLL Calibration . . . 459
8.3.4.1 Overview of DLL calibration process . . . 460
8.3.4.2 DLL Calibration flow . . . 460
8.3.5 DIMM Requirements . . . 460
8.3.6 Addressing . . . 461
8.3.7 Interface Between DDR and the Coherence Controller (COH) . . . 463
8.4 DDI Section . . . 463
8.4.1 Overview . . . 463
8.4.2 Request Path . . . 467
8.4.3 Read Shoot Down . . . 467
8.4.4 Data Path . . . 467
8.4.5 Requests to non-existent memory . . . 468
8.4.6 Powerdown . . . 468
8.4.7 Read Time-Out . . . 468
8.4.8 Registers and Definitions . . . 468
8.4.8.1 R_DdrxDdcDdpSoftReset - Soft Reset for DDC and DDP . . . 469
8.4.8.2 R_DdrxDdcMemCfg1 - Memory Controller Configuration Register 1 . . . 469
8.4.8.3 R_DdrxDdcMemCfg2 - Memory Controller Configuration Register 2 . . . 470
8.4.8.4 R_DdrxDdcMemCfg3 - Memory Controller Configuration Register 3 . . . 471
8.4.8.5 R_DdrxDdcMemCfg4 - Memory Controller Configuration Register 4 . . . 472
8.4.8.6 R_DdrxDdcMemCfg5 - Memory Controller Configuration Register 5 . . . 473
8.4.8.7 R_DdrxDdcMemCfg6 - Memory Controller Configuration Register 6 . . . 473
8.4.8.8 R_DdrxDdcMemCfg7 - Memory Controller Configuration Register 7 . . . 475
8.4.8.9 R_DdrxDdcDIMMODT - Memory Controller ODT Selection Matrix Configuration . 475
8.4.8.10 R_DdrxDdpODT - On-Die-Termination resistance value on ICE9 DDR2-I/O PADs during reads . . . 475
8.4.8.11 R_DdrxDIMMSize - Size of the DIMM this DDR unit instance is interfacing with. . . 476
8.4.8.12 R_DdrxDdiMifCfg1 - Memory Interface Configuration Register 1 . . . 476
8.4.8.13 R_DdrxDdiMifCfg2 - Memory Interface Configuration Register 2 . . . 477
8.4.8.14 R_DdrxPhyCfg1 - PHY Interface Configuration Register 1 . . . 478
8.4.8.15 R_DdrxPhyCfg2 - PHY Interface Configuration Register 2 . . . 479
8.4.8.16 R_DdrxPhyCfg3 - PHY Interface Configuration Register 3 . . . 483
8.4.8.17 R_DdrxDdpDLLLane0 - PHY Read Lane 0 DLL Configuration Register . . . 484
8.4.8.18 R_DdrxDdpDLLLane1 - PHY Read Lane 1 DLL Configuration Register . . . 484
8.4.8.19 R_DdrxDdpDLLLane2 - PHY Read Lane 2 DLL Configuration Register . . . 485
8.4.8.20 R_DdrxDdpDLLLane3 - PHY Read Lane 3 DLL Configuration Register . . . 485
8.4.8.21 R_DdrxDdpDLLLane4 - PHY Read Lane 4 DLL Configuration Register . . . 486
8.4.8.22 R_DdrxDdpDLLLane5 - PHY Read Lane 5 DLL Configuration Register . . . 486
8.4.8.23 R_DdrxDdpDLLLane6 - PHY Read Lane 6 DLL Configuration Register . . . 487
8.4.8.24 R_DdrxDdpDLLLane7 - PHY Read Lane 7 DLL Configuration Register . . . 487
8.4.8.25 R_DdrxDdpDLLLane8 - PHY Read Lane 8 DLL Configuration Register . . . 488
8.4.8.26 R_DdrxDdpDLLReset - PHY DLL Reset . . . 488

8.4.8.27 R_DdrxDdpCKReset - Reset for CK clock outputs to DIMM . . . 488
8.4.8.28 R_DdrxDddRdDelay . . . 489
8.4.8.29 R_DdrxDdiMemLoopBack . . . 489
8.4.8.30 R_DdrxDdiRdPathRst . . . 490
8.4.8.31 R_DdrxDdiRdTimeOut . . . 490
8.4.8.32 R_DdrxDdpCalReset . . . 490
8.4.8.33 R_DdrxDdpCalError . . . 491
8.4.8.34 R_DdrxDdpCalEnable . . . 491
8.4.8.35 R_DdrxDdpCalCounter . . . 492
8.4.8.36 R_DdrxDdpImpedCal . . . 492
8.4.8.37 R_DdrxDdpDataDrv . . . 494
8.4.8.38 R_DdrxDdpDQSDrv . . . 495
8.4.8.39 R_DdrxDdpCmdDrv . . . 495
8.4.8.40 R_DdrxDdiPHYWrptrCopy - This read only CSR is intended to be used for debugging only. The values only become valid after the last outstanding read has completed. The pointer is gray coded. When all outstanding reads have completed, the value of the R_DdrxDdiPHYWrptrCopy is expected to be 0001, 0111, 1101, or 1011. . . . 496
8.4.8.41 R_DdrxDdpHoldFix - This register has be included as a preventive measure. If it turns out that there are hold time problems with the sending of cmd/addr signals to the DIMM. Setting bits in this register muxes in delay elements to add additional hold time margin. . . . 496
8.4.8.42 R_DdrxDdpHighSpeedTest - This CSR is only intended for use during chip testing, where a tester is acting as a DIMM. . . . 496
8.4.8.43 R_DdrxDdiECCCaptureEnable . . . 497
8.4.8.44 R_DdrxDdiRdECCCapture0 . . . 497
8.4.8.45 R_DdrxDdiRdECCCapture1 . . . 498
8.4.9 Register Allocation . . . 498
8.4.9.1 Ddr0 . . . 498
8.4.9.2 Ddr1 . . . 498
8.4.10 Vregs_End_Of_Decl . . . 499
8.4.11 DDR Performace Events . . . 499
8.5 DDC Section - DDR2 SDRAM Controller IP Block . . . 499
8.6 DDD Section - Datapath interface to PHY . . . 499
8.7 DDP Unit - DDR2 SDRAM PHY IP Block . . . 500
8.7.1 Overview . . . 500
8.7.2 Clocks . . . 500
8.7.3 Address and Command Path . . . 500
8.7.4 Write Path . . . 500
8.7.5 Read Path . . . 500
8.7.6 DLLs . . . 501
8.7.6.1 DLL Master Adjustment . . . 501
8.7.6.2 DLL range calculations for Slave0 (DQS preamble enable DLL to match board trace length to memory) . . . 501
8.7.6.3 DLL range calculations for Slave1 (DQS 1/4 cycle delay DLL) . . . 501
8.7.7 I/O pads . . . 502
8.7.7.1 Impedence Calibration . . . 502

9 Counters, Performance Counters, & OCLA Overview 503
9.1 What's Available . . . 503
9.2 Status Bits . . . 503
9.3 Counters . . . 504
9.4 CPU Performance Counters . . . 504
9.5 SCB Performance Counters . . . 504
9.5.1 Ordinary Counting with SCB Performance Counters . . . 505
9.5.2 Statistical Counting with SCB Performance Counters . . . 505

9.6 OCLA . . . 506
9.6.1 OCLA Driving an External Pin . . . 506
9.6.2 OCLA as a Counter . . . 506
9.6.3 OCLA as a Times-of-Occurance Recorder . . . 507
9.6.4 OCLA as a Logic Analyzer . . . 507

10 Serial Configuration Bus 509
10.1 Overview . . . 509
10.2 Specifications . . . 509
10.3 Differences, Bugs, and Enhancements . . . 509
10.3.1 Product and Chip Pass Differences . . . 509
10.3.2 Known Bugs and Possible Enhancements . . . 510
10.4 Block Diagram . . . 510
10.5 SCB Master Ports . . . 510
10.6 SCB Slave Ports . . . 511
10.7 Custom/Large Structures . . . 512
10.8 I/O Operations . . . 512
10.8.1 No responder . . . 513
10.8.2 Approximate Latency . . . 513
10.8.3 Software Notes . . . 513
10.9 SysChain Interface . . . 513
10.9.1 SysChain Access Requirements . . . 513
10.9.2 SysChain SCB Write . . . 513
10.9.3 SysChain SCB Read . . . 514
10.10 Performance Counting . . . 514
10.10.1 True Counting . . . 514
10.10.2 Statistical Counting . . . 514
10.10.3 Counts Causing Interrupts . . . 515
10.10.4 OCLA Triggering . . . 515
10.10.5 Events from OCLA . . . 515
10.10.6 Arbitration . . . 515
10.10.7 Software Notes . . . 515
10.10.8 Writing while Counting . . . 516
10.11 Connecting to SCBS . . . 516
10.11.1 List of Slaves . . . 516
10.11.2 Slave I/O Transactions . . . 516
10.11.3 Slave Performance Counting Interface . . . 516
10.12 SCB Internals . . . 517
10.12.1 PMI Interface . . . 517
10.12.2 SCB Bus Protocol . . . 517
10.12.3 ICE9 Bit Sequence . . . 518
10.12.4 TWC9+ Bit Sequence . . . 518
10.12.5 Commands . . . 519
10.12.5.1 Idle . . . 519
10.12.5.2 Reset . . . 519
10.12.5.3 AddrH . . . 519
10.12.5.4 Write . . . 519
10.12.5.5 Read . . . 520
10.12.5.6 Count . . . 520
10.13 Chip Reset . . . 520
10.14 Registers and Definitions . . . 520
10.14.1 Package Attributes . . . 520
10.14.2 Definitions . . . 520
10.14.3 Command Enumerations . . . 521
10.14.4 Data Ack Enumerations . . . 521
10.14.5 SCB Performance Events . . . 521

10.14.6 Chip Revision Register 522
10.14.7 Chip Number Register 522
10.14.8 Chip Null Subcomponent Register 522
10.14.9 Chip Speed Register 523
10.14.10 General Purpose IO Register 524
10.14.11 LED Register 524
10.14.12 Attention Chip Register 524
10.15 Debug Attention Interrupt Register 526
10.16 Debug Interrupt Register 526
10.17 Performance Counting Registers 527
10.17.1 Interrupt Register 527
10.17.2 Interrupt Mask Register 528
10.17.3 Interrupt Request Register 528
10.17.4 Performance Control Register 529
10.17.5 Performance Histogram Register 529
10.17.6 Performance Bucket Number Register 530
10.17.7 Performance Enable Register 530
10.17.8 Performance Status Register 531
10.17.9 Performance Bucket Configuration 531
10.17.10 Performance Count Ram 532

11 On Chip Logic Analyzer 535
11.1 Overview 535
11.2 Differences, Bugs, and Enhancements 535
11.2.1 Product and Chip Pass Differences 535
11.2.2 Known Bugs 536
11.2.3 Possible Enhancements 536
11.3 Description 538
11.4 Package Attributes 539
11.5 LAC Signals and Innards 540
11.5.1 What LAC Does 540
11.5.2 LAC Innards 540
11.5.2.1 LAC to SCB-Performance-Counters 540
11.5.2.2 SCB-Performance-Counters to LAC 541
11.5.2.3 LAC Operation Codes 541
11.5.2.4 Be Sure To Shut Off CollectTrace 541
11.5.3 LAC Registers 542
11.5.3.1 The Control Register 542
11.5.3.2 The Delay Registers 542
11.5.3.3 The Aggregate Mask Registers 542
11.5.3.4 The Aggregate Match Registers 543
11.5.3.5 The Initial Counter Value Registers 544
11.5.3.6 The Current Counter Value Registers 544
11.5.3.7 The FSM RAM 544
11.5.4 LAC Signals 545
11.6 Collector Blocks (CTBs) in general 545
11.6.1 CTB Innards 545
11.6.1.1 The Control Unit and Muxes 547
11.6.1.2 The WT Addr Register 547
11.6.1.3 The Dead Cycle Counter 547
11.6.1.4 A Dead Cycle Counter Bug 547
11.6.1.5 The Trace RAM 547
11.6.1.6 When Can You Read CTB Contents? 547
11.6.1.7 Do You Need To Shut-Off CollectTrace? 548
11.6.2 Registers 548
11.6.2.1 The Collection Control Register 548

11.6.2.2 The RAM Lowbits . . . 549
11.6.2.3 The RAM Highbits . . . 549
11.6.2.4 The Write Address . . . 549
11.6.3 CTB Signals . . . 550
11.7 Hints for Using Collector Blocks . . . 550
11.7.1 Collecting the Event You Triggered On . . . 550
11.8 Vector Trigger Blocks (TRBVs) in general . . . 551
11.8.1 SCB Performance Counter Connections . . . 552
11.8.2 Registers . . . 552
11.8.2.1 The Trigger Control Register . . . 552
11.8.2.2 The Trigger Mask Registers . . . 552
11.8.2.3 The Trigger Match Registers . . . 553
11.8.3 TRBV Signals . . . 553
11.9 Codeword Trigger Blocks (TRBCs) in general . . . 553
11.9.1 SCB Performance Counter Connections . . . 555
11.9.2 Registers . . . 555
11.9.2.1 The Trigger Control Register . . . 555
11.9.2.2 The Trigger Table Registers . . . 556
11.9.2.3 The Qualifier Table Registers . . . 556
11.9.3 TRBC Signals . . . 556
11.10 Hints for Using Trigger Blocks . . . 557
11.10.1 Using CodeValid Signals . . . 557
11.10.2 Trigger Clock Domains . . . 557
11.10.3 Uses for the Delay Registers . . . 557
11.10.3.1 Aligning Mis-Aligned Signals From Same Trigger Block . . . 557
11.10.3.2 Aligning CodeValid or Qualifier with Other Triggers in a Trigger Block . . . 557
11.10.3.3 Aligning Triggers from Different Trigger Blocks . . . 557
11.10.3.4 Provide Bigger Window for Coinciding Events . . . 558
11.11 OCLA in use – PSx (Processor Segments) . . . 558
11.11.0.5 Location of OCLA-PSx Blocks and Signals . . . 558
11.11.1 PSx Triggers . . . 558
11.11.1.1 Processor Segment Trigger Mux 0 . . . 558
11.11.1.2 Processor Segment Trigger Mux 1 . . . 559
11.11.1.3 Processor Segment Trigger Mux 2 . . . 559
11.11.1.4 Processor Segment Trigger Mux 3 . . . 560
11.11.2 PSx Collectors . . . 561
11.11.2.1 PSx Input Collectors Qualifying Triggers . . . 561
11.11.2.2 PSx Input Collector Mux 0 . . . 561
11.11.2.3 PSx Input Collector Mux 1 . . . 562
11.11.2.4 PSx Input Collector Mux 2 . . . 562
11.11.2.5 PSx Input Collector Mux 3 . . . 563
11.11.2.6 PSx Input Collector Mux 4, 5, 6, 7 . . . 564
11.12 OCLA in use – COHx . . . 564
11.12.0.7 COHx Trigger and Collector Enabling . . . 564
11.12.1 COHx Triggers . . . 564
11.12.1.1 COHx Codeword Trigger Mux 0: Trigger on incoming command/source/data-op + tag-results + orc/wbc hit . . . 564
11.12.1.2 COHx Codeword Trigger Mux 1: Trigger on ORC/WBC behavior + incoming command . . . 565
11.12.1.3 COHx Codeword Trigger Mux 2: Trigger on the DDR Interface . . . 565
11.12.1.4 COHx Codeword Trigger Mux 3: Trigger on an Incoming Address . . . 566
11.12.2 COHx Collectors . . . 566
11.12.2.1 Cohx Input Collectors Qualifying Triggers . . . 566
11.12.2.2 Cohx Input Collector Mux 0 . . . 566
11.12.2.3 Cohx Input Collector Mux 1 . . . 567
11.12.2.4 Cohx Input Collector Mux 2 . . . 567

11.12.2.5 Cohx Input Collector Mux 3 568
11.12.2.6 Cohx Input Collector Mux 4 568
11.12.2.7 Cohx Input Collector Mux 5, or 6 569
11.12.2.8 Cohx Input Collector Mux 7 569
11.13 OCLA in use – FSW 569
11.13.1 FSW Triggers 569
11.13.1.1 FSW Codeword Trigger Block Inputs 569
11.13.1.2 FSW Input Vector Trigger (Mux 0) 570
11.13.1.3 FSW Input Vector Trigger (Mux 1) 570
11.13.1.4 FSW Input Vector Trigger Mux 2 570
11.13.1.5 FSW Input Vector Trigger Mux 3 571
11.13.1.6 FSW Input Vector Trigger Mux 4 571
11.13.1.7 FSW Output Vector Trigger Mux 0 571
11.13.1.8 FSW Output Vector Trigger Mux 1 572
11.13.1.9 FSW Output Vector Trigger Mux 2 572
11.13.1.10 FSW Output Vector Trigger Mux 3 572
11.13.1.11 FSW Output Vector Trigger Mux 4 573
11.13.2 FSW Collectors 573
11.13.2.1 FSW Input Collectors Qualifying Triggers 573
11.13.2.2 FSW Input Collector Mux 0 573
11.13.2.3 FSW Input Collector Mux 1 574
11.13.2.4 FSW Input Collector Mux 2 574
11.13.2.5 FSW Input Collector Mux 3 574
11.13.2.6 FSW Input Collector Mux 4 574
11.13.2.7 FSW Input Collector Mux 5, 6, 7 575
11.13.2.8 FSW Output Collectors Qualifying Triggers 575
11.13.2.9 FSW Output Collector Mux 0 575
11.13.2.10 FSW Output Collector Mux 1 575
11.13.2.11 FSW Output Collector Mux 2 575
11.13.2.12 FSW Output Collector Mux 3 576
11.13.2.13 FSW Output Collector Mux 4 576
11.13.2.14 FSW Output Collector Mux 5, 6, 7 576
11.14 OCLA in use – DMA 576
11.14.1 DMA Triggers 576
11.14.1.1 DMA Codeword Triggers 576
11.14.1.2 DMA Vector Trigger Inputs (Mux 0) 577
11.14.1.3 DMA Vector Trigger Inputs (Mux 1) 578
11.14.1.4 DMA Vector Trigger Inputs (Mux 2) 578
11.14.1.5 DMA Vector Trigger Inputs (Mux 3) 579
11.14.2 DMA Collector 579
11.14.2.1 DMA Input Collectors Qualifying Triggers 579
11.14.2.2 DMA Input Collector Mux 0 580
11.14.2.3 DMA Input Collector Mux 1 581
11.14.2.4 DMA Input Collector Mux 2 581
11.14.2.5 DMA Input Collector Mux 3 582
11.14.2.6 DMA Input Collector Mux 4, 5, 6, 7 582
11.15 OCLA in use – PMI 582
11.15.1 PMI/PCI/BBS Triggers 582
11.15.1.1 "TrbcPmi" PMI CSW Bus Stop Codeword Triggers 582
11.15.1.2 "TrbcPmii" PMI Internal Signal Codeword Triggers 583
11.15.2 PMI/PCI/BBS Collector 583
11.15.2.1 PMI Input Qualifying Triggers 583
11.15.2.2 PMI Input Collector Mux 0 584
11.15.2.3 PMI Input Collector Mux 1 584
11.15.2.4 PMI Input Collector Mux 2 585
11.15.2.5 PMI Input Collector Mux 3 585

11.15.2.6 PMI Input Collector Mux 4 . . . 585
11.15.2.7 PMI Input Collector Mux 5 . . . 586
11.15.2.8 PMI Input Collector Mux 6 . . . 586
11.15.2.9 PMI Input Collector Mux 7 . . . 586
11.16 Register Address Ranges . . . 587
11.16.1 TrbcPs0 . . . 587
11.16.2 TrbcPs1 . . . 587
11.16.3 TrbcPs2 . . . 587
11.16.4 TrbcPs3 . . . 587
11.16.5 TrbcPs4 . . . 587
11.16.6 TrbcPs5 . . . 588
11.16.7 TrbcPs6 . . . 588
11.16.8 TrbcPs7 . . . 588
11.16.9 TrbcPs8 . . . 588
11.16.10 TrbcPs9 . . . 588
11.16.11 TrbcDma . . . 589
11.16.12 TrbvDma . . . 589
11.16.13 TrbcPmi . . . 589
11.16.14 TrbcPmii . . . 589
11.16.15 TrbcCoho . . . 589
11.16.16 TrbcCohe . . . 589
11.16.17 TrbvFswo . . . 590
11.16.18 TrbvFswi . . . 590
11.16.19 TrbcFsw . . . 590
11.16.20 CtbPs0 . . . 590
11.16.21 CtbPs1 . . . 590
11.16.22 CtbPs2 . . . 590
11.16.23 CtbPs3 . . . 591
11.16.24 CtbPs4 . . . 591
11.16.25 CtbPs5 . . . 591
11.16.26 CtbPs6 . . . 591
11.16.27 CtbPs7 . . . 591
11.16.28 CtbPs8 . . . 592
11.16.29 CtbPs9 . . . 592
11.16.30 CtbDma . . . 592
11.16.31 CtbPmi . . . 592
11.16.32 CtbCoho . . . 592
11.16.33 CtbCohe . . . 592
11.16.34 CtbFswi . . . 593
11.16.35 CtbFswo . . . 593
11.17 OCLA Programming Suggestions . . . 593
11.17.1 Ready-To-Use OCLA Scripts . . . 593
11.17.2 Example Code for OCLA . . . 593
11.17.3 Use Our Examples on a Real Machine . . . 593
11.17.4 Create Your Own Counter . . . 594
11.17.4.1 You might prefer SCB Performance Counters . . . 594
11.17.5 Defensive Programming . . . 594
11.17.6 CTB stuck-at-full . . . 595
11.17.7 Shutting-Off CollectTrace . . . 595
11.17.7.1 Why would CollectTrace be Left ON? . . . 595
11.17.7.2 Why is CollectTrace ON a Problem? . . . 595
11.17.7.3 Is CollectTrace ON? . . . 596
11.17.7.4 How to Read CTB Contents While CollectTrace is ON . . . 596
11.17.7.5 Fastest Way To Shut Off CollectTrace in Ice9A . . . 596

12 Clocking, ECC, Test Logic, Reset, and Initialization 597
12.1 Overview . . . 597
12.2 Differences, Bugs, and Enhancements . . . 597
12.2.1 Product and Chip Pass Differences . . . 597
12.2.2 Known Bugs and Possible Enhancements . . . 598
12.3 Clock generation and distribution . . . 598
12.3.1 Goals and Features . . . 598
12.3.2 Sys_clk distribution tree . . . 599
12.3.3 Clock Generation in ICE9 . . . 600
12.3.4 PCIe clocking . . . 600
12.3.5 Block diagram of PLL_AB . . . 601
12.3.5.1 Bypass mode in PLL_AB . . . 603
12.3.6 Implementation of PLL_AB . . . 603
12.4 General ECC strategy . . . 605
12.4.1 ECC Control Register descriptions: . . . 605
12.4.1.1 ECC_Mode_Register[1:0] (associated with ECC correction) . . . 605
12.4.1.2 ECC_Drive_Bad_Data_Register[1:0] (associated with ECC generation) . . . 605
12.4.2 ECC Status Register Descriptions . . . 605
12.4.2.1 ECC_Error_Status_Register[2:0] (associated with ECC correction) . . . 605
12.4.2.2 ECC_Error_Address_Register[x:0] - x depends on the size of address space (associated with ECC correction) . . . 606
12.4.2.3 ECC_Error_Syndrom_Register[7:0] (associated with ECC correction) . . . 606
12.4.3 ECC Implementation & Test considerations . . . 606
12.4.3.1 Compiled memories with Synchronous Write Through (SWT) mode . . . 606
12.4.3.2 Compiled memories with Asynchronous Write Through (AWT) and no Synchronous Write Through (SWT) . . . 606
12.5 DFT and Test Support . . . 607
12.5.1 Boundary scan (normal mode) . . . 608
12.5.2 Stuck-at Scan (test mode 16) . . . 608
12.5.3 Transition Fault Scan (test_mode 17) . . . 608
12.5.4 PLL Test (test mode 18) . . . 610
12.5.5 DDR ODT & Drive Strength Parametric Test (test mode 19) . . . 610
12.5.6 Memory BIST and Repair (test mode 0, 20) . . . 610
12.5.7 DDR Functional Test (test modes 0, 21) . . . 610
12.5.8 Slow DDR DLL Test (test mode 22) (whether all DLL tests will be used in mfg. test is still open) . . . 610
12.5.8.1 DLL low speed test 1 (DLL vendor recommended) . . . 610
12.5.8.2 DLL low speed test 2 (DLL vendor recommended) . . . 611
12.5.9 Fast DDR DLL Test (test mode 23) (whether all DLL tests will be used in mfg. test is still open) . . . 611
12.5.9.1 DLL High Speed Test 1 . . . 611
12.5.9.2 DLL Functional Slave Test . . . 611
12.5.10 PCI Functional Tests (test modes 0, 24, 25, or 26) . . . 612
12.5.11 Fabric Transceiver Functional Test (test modes 27, 28) . . . 612
12.6 SysChain . . . 612
12.6.1 SysChain Ordering Rules . . . 613
12.6.2 Vregs Package . . . 613
12.6.3 SysChain TAP Constants . . . 613
12.6.4 SysChain TAP Enumeration . . . 613
12.6.5 System TAP Instructions . . . 615
12.6.6 System TAP Instruction Register . . . 618
12.6.7 System TAP Instruction Register for TWC9 . . . 618
12.6.8 Device Identification Register . . . 619
12.6.9 PLL Control Register . . . 619
12.6.10 Reset Control Register . . . 620
12.6.11 Memory Init Register . . . 622

12.6.12 Processor Debug Interrupt Register . . . 623
12.6.13 SMS BIST Contol Register . . . 623
12.6.14 Serial Configuration Bus Interface Register . . . 624
12.6.15 MSP-Hosted Node Attention Register . . . 625
12.6.16 External JTAG Chains . . . 626
12.7 Global reset . . . 626
12.8 Boot Timeline . . . 628
12.8.1 SSP Boot Timeline . . . 628
12.8.2 MSP Boot Timeline . . . 628
12.8.3 Pre-DRAM Boot Timeline . . . 630
12.8.4 DRAM Boot Timeline . . . 631
12.8.5 Kernel Boot Timeline . . . 632
12.8.6 Booting the Fabric Switch and Links . . . 632
12.8.7 Booting the DMA Engine . . . 632
12.8.8 Rebooting with Fabric Still Up . . . 633

13 PCI Express Subsystem **635**
13.1 Overview . . . 635
13.2 Differences, Bugs, and Enhancements . . . 635
13.2.1 Product and Chip Pass Differences . . . 635
13.2.2 Known Bugs and Possible Enhancements . . . 636
13.3 Internal Structure . . . 636
13.4 Known Bugs and Enhancements . . . 636
13.5 Process Requirements . . . 636
13.6 Application Layer and the PMI . . . 636
13.6.1 The Requestor Unit REQ . . . 638
13.6.1.1 REQ Memory Read Request Handling . . . 638
13.6.1.2 REQ Memory Write Request Handling . . . 639
13.6.1.3 REQ IO Read Request Handling . . . 639
13.6.1.4 REQ IO Write Request Handling . . . 639
13.6.1.5 REQ Configuration Read Request Handling . . . 639
13.6.1.6 REQ Configuration Write Request Handling . . . 639
13.6.1.7 REQ Sub-blocks . . . 640
13.6.1.8 REQ Exception Handling . . . 640
13.6.1.9 RC Config Register Access . . . 640
13.6.2 The Completer Unit CMP . . . 640
13.6.2.1 Memory Write Operation . . . 641
13.6.2.2 Memory Read Operation . . . 641
13.6.2.3 Message Signalled Interrupts . . . 644
13.6.3 The Control/Status Widget CSI . . . 644
13.6.3.1 The CSW Interface CIF . . . 644
13.6.3.2 The Wishbone Interface WBI . . . 644
13.6.3.3 The RC Register Interface DBI . . . 646
13.6.3.4 The Phy Interface CRI . . . 646
13.6.3.5 The PMI Register Block CIN . . . 646
13.6.3.6 CSI Exception Handling . . . 646
13.6.4 The Command/Address Multiplexer CMX . . . 647
13.6.5 The Data Multiplexer DMX . . . 647
13.7 Valid CSW Operations . . . 647
13.8 Valid PCI Operations . . . 647
13.9 Ordering Rules . . . 648
13.10 Auxiary PCI Signals . . . 648
13.10.1 PERST# output . . . 648
13.10.2 MPWRGD# input . . . 648
13.10.3 PWRFLT# input . . . 648
13.10.4 PWREN# output . . . 648

13.10.5 PRSNT# input . . . 649
13.10.6 ATNLED output . . . 649
13.10.7 PWRLED output . . . 649
13.11 Definitions . . . 649
13.11.1 PCI Type Enumerations . . . 649
13.11.2 PCI Format Enumerations . . . 649
13.11.3 PCI Completion Status Enumerations . . . 649
13.11.4 PCI Completion State Machine State Enumerations . . . 650
13.11.5 PCI Block Write State Machine State Enumerations . . . 650
13.11.6 PCI Block Read State Machine State Enumerations . . . 650
13.11.7 PMI Request Result Enumerations . . . 650
13.11.8 Pmi Events . . . 651
13.12 PCI Express Root Complex Registers . . . 652
13.12.1 Device/Vendor ID Register . . . 652
13.12.2 Command and Status Register . . . 652
13.12.3 RevID, Class Code Register . . . 653
13.12.4 Cache Line Size, BIST etc register . . . 654
13.12.5 Base Address Register 0 . . . 654
13.12.6 Base Address Register 1 . . . 654
13.12.7 Bus Number Register . . . 655
13.12.8 I/O Base/Limit, and Secondary Status Register . . . 655
13.12.9 Non-Prefetchable Memory Base and Limit Register . . . 656
13.12.10 Prefetchable Memory Base and Limit Register . . . 657
13.12.11 Prefetchable Memory Upper Base Register . . . 657
13.12.12 Prefetchable Memory Upper Limit Register . . . 657
13.12.13 I/O Base and Limit Upper Register . . . 658
13.12.14 Capability Pointer Register . . . 658
13.12.15 Expansion ROM Register . . . 658
13.12.16 Bridge Control Register . . . 659
13.12.17 PCI Power Management Capabilities Register . . . 659
13.12.18 PCI Power Management Control Register . . . 660
13.12.19 MSI Capabilities Register . . . 660
13.12.20 MSI Address Register . . . 661
13.12.21 MSI Upper Address/Data Register . . . 661
13.12.22 MSI Data Register . . . 661
13.12.23 PCI Express Capabilities Register 0 . . . 662
13.12.24 PCI Express Capabilities Register 1 . . . 662
13.12.25 Device Control/Status Register . . . 663
13.12.26 Link Capabilities Register . . . 663
13.12.27 Link Control/Status Register . . . 664
13.12.28 Slot Capabilities Register . . . 665
13.12.29 Slot Control/Status Register . . . 665
13.12.30 Root Control Register . . . 666
13.12.31 Root Status Register . . . 667
13.12.32 Advanced Error Reporting Enhanced Capability Header Register . . . 667
13.12.33 Advanced Error Reporting Uncorrectable Error Status Register . . . 667
13.12.34 Uncorrectable Error Mask Register . . . 668
13.12.35 Uncorrectable Severity Register . . . 669
13.12.36 Correctable Error Status Register . . . 669
13.12.37 Correctable Error Mask Register . . . 670
13.12.38 Advanced Error Capabilities Control Register . . . 670
13.12.39 Advanced Error Capabilities/Header Log Register (1st Dword) . . . 671
13.12.40 Header Log Register (2nd Dword) . . . 671
13.12.41 Header Log Register (3rd Dword) . . . 671
13.12.42 Header Log Register (4th Dword) . . . 672
13.12.43 Root Error Command Register . . . 672

13.12.44 Root Error Status Register 672
13.12.45 Root Error Source Identification Register 673
13.13 PMI Control and Status Registers 674
13.13.1 Core Control Register 674
13.13.2 PMI Interrupt Summary Register 674
13.13.3 PMI Interrupt Enable Register 676
13.13.4 LED Blink Rate Register 678
13.13.5 Send Unlock Message Register 678
13.13.6 Send Turnoff Message Register 678
13.13.7 Link Status Register 678
13.13.8 Root-Complex Debug Info 679
13.13.9 Force Ecc Error Register 679
13.13.10 CSI Ecc Error Register 680
13.13.11 CSI Address Error Register 680
13.13.12 DBI 64bit Access Error Register 681
13.13.13 CSI Wishbone Timeout Error Register 681
13.13.14 REQ Ecc Error Register 682
13.13.15 REQ Completion Error Register 682
13.13.16 SYC CSW Ecc Error Register 683
13.13.17 CCW CSW Ecc Error Register 683
13.13.18 CCW SYC Ecc Error Register 684
13.13.19 MSI Address Register 684
13.13.20 Wishbone Timeout Value Register 685
13.13.21 VSM Request Double Word 1 and 2 Register 685
13.13.22 VSM Request Double Word 3 and 4 Register 685
13.13.23 VMI Request Data Register 685
13.13.24 Received Vendor Message Double Word 1 and 2 Register 686
13.13.25 Received Vendor Message Double Word 3 and 4 Register 686
13.13.26 Received Vendor Message Payload Register 686
13.14 PCI Express Phy Registers 687
13.14.1 Less Than Limit Compare Point Register 687
13.14.2 Greater Than Limit Compare Point Register 687
13.14.3 Compare/Scratch Value Mask Register 687
13.14.4 Scratch Space Control Register 688
13.14.5 Scratch Register Comparisons To Limits Results Register 688
13.14.6 Number Of Samples To Count Register 688
13.14.7 Scope Counting Results Register 689
13.14.8 Support DAC Values And Controls Register 689
13.14.9 Resistor Tuning Controls Register 689
13.14.10 ADC Process Results Register 690
13.14.11 Current MPLL Phase Selector Value Register 690
13.14.12 JTAG Chip ID Register (Lower 16 Bits) 690
13.14.13 JTAG Chip ID Register (Upper 16 Bits) 691
13.14.14 Frequency Control Inputs Status Register 691
13.14.15 Various Control Inputs Status Register 691
13.14.16 Level Control Inputs Status Register 692
13.14.17 Creg Control I/O Status Register 692
13.14.18 Frequency Control Inputs Override Register 693
13.14.19 Various Control Inputs Override Register 693
13.14.20 Level Control Inputs Override Register 694
13.14.21 Creg Control I/O Override Register 694
13.14.22 MPLL Controls Register 695
13.14.23 MPLL Test Controls Register 695
13.14.24 Transmit Control Inputs Status Register (Lane 0) 695
13.14.25 Receiver Control Inputs Status Register (Lane 0) 696
13.14.26 Output Signals Status Register (Lane 0) 696

13.14.27 Transmitter Control Inputs Override Register (Lane 0) . . . 697
13.14.28 Receiver Control Inputs Override Register (Lane 0) . . . 697
13.14.29 Output Signals Override Register (Lane 0) . . . 697
13.14.30 Debug Control Register (Lane 0) . . . 698
13.14.31 Pattern Generator Controls Register (Lane 0) . . . 698
13.14.32 Pattern Matcher Controls Register (Lane 0) . . . 698
13.14.33 Pattern Match Error Counter Register (Lane 0) . . . 699
13.14.34 Current Phase Selector Value. Register (Lane 0) . . . 699
13.14.35 Current Frequency Integrator Value. Register (Lane 0) . . . 700
13.14.36 Scope Control Register (Lane 0) . . . 700
13.14.37 Recovered Domain Receiver Control Register (Lane 0) . . . 700
13.14.38 Receiver Debug Register (Lane 0) . . . 701
13.14.39 RX Control Register (Lane 0) . . . 701
13.14.40 RX ATB Register (Lane 0) . . . 702
13.14.41 8 Bit Programming Register (Lane 0) . . . 702
13.14.42 10 Bit Programming Register (Lane 0) . . . 703
13.14.43 10 Bit Programming Register (Lane 0) . . . 703
13.14.44 TX ATB Control Register (Set 1) (Lane 0) . . . 704
13.14.45 TX ATB Control Register (Set 2) (Lane 0) . . . 704
13.14.46 TX POWER STATE Control Register (Lane 0) . . . 705
13.14.47 Transmit Control Inputs Status Register (Lane 1) . . . 705
13.14.48 Receiver Control Inputs Status Register (Lane 1) . . . 706
13.14.49 Output Signals Status Register (Lane 1) . . . 706
13.14.50 Transmitter Control Inputs Override Register (Lane 1) . . . 707
13.14.51 Receiver Control Inputs Override Register (Lane 1) . . . 707
13.14.52 Output Signals Override Register (Lane 1) . . . 707
13.14.53 Debug Control Register (Lane 1) . . . 708
13.14.54 Pattern Generator Controls Register (Lane 1) . . . 708
13.14.55 Pattern Matcher Controls Register (Lane 1) . . . 708
13.14.56 Pattern Match Error Counter Register (Lane 1) . . . 709
13.14.57 Current Phase Selector Value. Register (Lane 1) . . . 709
13.14.58 Current Frequency Integrator Value. Register (Lane 1) . . . 710
13.14.59 Scope Control Register (Lane 1) . . . 710
13.14.60 Recovered Domain Receiver Control Register (Lane 1) . . . 710
13.14.61 Receiver Debug Register (Lane 1) . . . 711
13.14.62 RX Control Register (Lane 1) . . . 711
13.14.63 RX ATB Register (Lane 1) . . . 712
13.14.64 8 Bit Programming Register (Lane 1) . . . 712
13.14.65 10 Bit Programming Register (Lane 1) . . . 713
13.14.66 10 Bit Programming Register (Lane 1) . . . 713
13.14.67 TX ATB Control Register (Set 1) (Lane 1) . . . 714
13.14.68 TX ATB Control Register (Set 2) (Lane 1) . . . 714
13.14.69 TX POWER STATE Control Register (Lane 1) . . . 715
13.14.70 Transmit Control Inputs Status Register (Lane 2) . . . 715
13.14.71 Receiver Control Inputs Status Register (Lane 2) . . . 716
13.14.72 Output Signals Status Register (Lane 2) . . . 716
13.14.73 Transmitter Control Inputs Override Register (Lane 2) . . . 717
13.14.74 Receiver Control Inputs Override Register (Lane 2) . . . 717
13.14.75 Output Signals Override Register (Lane 2) . . . 717
13.14.76 Debug Control Register (Lane 2) . . . 718
13.14.77 Pattern Generator Controls Register (Lane 2) . . . 718
13.14.78 Pattern Matcher Controls Register (Lane 2) . . . 718
13.14.79 Pattern Match Error Counter Register (Lane 2) . . . 719
13.14.80 Current Phase Selector Value. Register (Lane 2) . . . 719
13.14.81 Current Frequency Integrator Value. Register (Lane 2) . . . 720
13.14.82 Scope Control Register (Lane 2) . . . 720

13.14.83 Recovered Domain Receiver Control Register (Lane 2) 720
13.14.84 Receiver Debug Register (Lane 2) 721
13.14.85 RX Control Register (Lane 2) 721
13.14.86 RX ATB Register (Lane 2) 722
13.14.87 8 Bit Programming Register (Lane 2) 722
13.14.88 10 Bit Programming Register (Lane 2) 723
13.14.89 10 Bit Programming Register (Lane 2) 723
13.14.90 TX ATB Control Register (Set 1) (Lane 2) 724
13.14.91 TX ATB Control Register (Set 2) (Lane 2) 724
13.14.92 TX POWER STATE Control Register (Lane 2) 725
13.14.93 Transmit Control Inputs Status Register (Lane 3) 725
13.14.94 Receiver Control Inputs Status Register (Lane 3) 726
13.14.95 Output Signals Status Register (Lane 3) 726
13.14.96 Transmitter Control Inputs Override Register (Lane 3) 727
13.14.97 Receiver Control Inputs Override Register (Lane 3) 727
13.14.98 Output Signals Override Register (Lane 3) 727
13.14.99 Debug Control Register (Lane 3) 728
13.14.100 Pattern Generator Controls Register (Lane 3) 728
13.14.101 Pattern Matcher Controls Register (Lane 3) 728
13.14.102 Pattern Match Error Counter Register (Lane 3) 729
13.14.103 Current Phase Selector Value. Register (Lane 3) 729
13.14.104 Current Frequency Integrator Value. Register (Lane 3) 730
13.14.105 Scope Control Register (Lane 3) 730
13.14.106 Recovered Domain Receiver Control Register (Lane 3) 730
13.14.107 Receiver Debug Register (Lane 3) 731
13.14.108 RX Control Register (Lane 3) 731
13.14.109 RX ATB Register (Lane 3) 732
13.14.110 8 Bit Programming Register (Lane 3) 732
13.14.111 10 Bit Programming Register (Lane 3) 733
13.14.112 10 Bit Programming Register (Lane 3) 733
13.14.113 TX ATB Control Register (Set 1) (Lane 3) 734
13.14.114 TX ATB Control Register (Set 2) (Lane 3) 734
13.14.115 TX POWER STATE Control Register (Lane 3) 735
13.14.116 Transmit Control Inputs Status Register (Lane 4) 735
13.14.117 Receiver Control Inputs Status Register (Lane 4) 736
13.14.118 Output Signals Status Register (Lane 4) 736
13.14.119 Transmitter Control Inputs Override Register (Lane 4) 737
13.14.120 Receiver Control Inputs Override Register (Lane 4) 737
13.14.121 Output Signals Override Register (Lane 4) 737
13.14.122 Debug Control Register (Lane 4) 738
13.14.123 Pattern Generator Controls Register (Lane 4) 738
13.14.124 Pattern Matcher Controls Register (Lane 4) 738
13.14.125 Pattern Match Error Counter Register (Lane 4) 739
13.14.126 Current Phase Selector Value. Register (Lane 4) 739
13.14.127 Current Frequency Integrator Value. Register (Lane 4) 740
13.14.128 Scope Control Register (Lane 4) 740
13.14.129 Recovered Domain Receiver Control Register (Lane 4) 740
13.14.130 Receiver Debug Register (Lane 4) 741
13.14.131 RX Control Register (Lane 4) 741
13.14.132 RX ATB Register (Lane 4) 742
13.14.133 8 Bit Programming Register (Lane 4) 742
13.14.134 10 Bit Programming Register (Lane 4) 743
13.14.135 10 Bit Programming Register (Lane 4) 743
13.14.136 TX ATB Control Register (Set 1) (Lane 4) 744
13.14.137 TX ATB Control Register (Set 2) (Lane 4) 744
13.14.138 TX POWER STATE Control Register (Lane 4) 745

13.14.139 Transmit Control Inputs Status Register (Lane 5) . . . 745
13.14.140 Receiver Control Inputs Status Register (Lane 5) . . . 746
13.14.141 Output Signals Status Register (Lane 5) . . . 746
13.14.142 Transmitter Control Inputs Override Register (Lane 5) . . . 747
13.14.143 Receiver Control Inputs Override Register (Lane 5) . . . 747
13.14.144 Output Signals Override Register (Lane 5) . . . 747
13.14.145 Debug Control Register (Lane 5) . . . 748
13.14.146 Pattern Generator Controls Register (Lane 5) . . . 748
13.14.147 Pattern Matcher Controls Register (Lane 5) . . . 748
13.14.148 Pattern Match Error Counter Register (Lane 5) . . . 749
13.14.149 Current Phase Selector Value. Register (Lane 5) . . . 749
13.14.150 Current Frequency Integrator Value. Register (Lane 5) . . . 750
13.14.151 Scope Control Register (Lane 5) . . . 750
13.14.152 Recovered Domain Receiver Control Register (Lane 5) . . . 750
13.14.153 Receiver Debug Register (Lane 5) . . . 751
13.14.154 RX Control Register (Lane 5) . . . 751
13.14.155 RX ATB Register (Lane 5) . . . 752
13.14.156 8Bit Programming Register (Lane 5) . . . 752
13.14.157 10 Bit Programming Register (Lane 5) . . . 753
13.14.158 10 Bit Programming Register (Lane 5) . . . 753
13.14.159 TX ATB Control Register (Set 1) (Lane 5) . . . 754
13.14.160 TX ATB Control Register (Set 2) (Lane 5) . . . 754
13.14.161 TX POWER STATE Control Register (Lane 5) . . . 755
13.14.162 Transmit Control Inputs Status Register (Lane 6) . . . 755
13.14.163 Receiver Control Inputs Status Register (Lane 6) . . . 756
13.14.164 Output Signals Status Register (Lane 6) . . . 756
13.14.165 Transmitter Control Inputs Override Register (Lane 6) . . . 757
13.14.166 Receiver Control Inputs Override Register (Lane 6) . . . 757
13.14.167 Output Signals Override Register (Lane 6) . . . 757
13.14.168 Debug Control Register (Lane 6) . . . 758
13.14.169 Pattern Generator Controls Register (Lane 6) . . . 758
13.14.170 Pattern Matcher Controls Register (Lane 6) . . . 758
13.14.171 Pattern Match Error Counter Register (Lane 6) . . . 759
13.14.172 Current Phase Selector Value. Register (Lane 6) . . . 759
13.14.173 Current Frequency Integrator Value. Register (Lane 6) . . . 760
13.14.174 Scope Control Register (Lane 6) . . . 760
13.14.175 Recovered Domain Receiver Control Register (Lane 6) . . . 760
13.14.176 Receiver Debug Register (Lane 6) . . . 761
13.14.177 RX Control Register (Lane 6) . . . 761
13.14.178 RX ATB Register (Lane 6) . . . 762
13.14.179 8Bit Programming Register (Lane 6) . . . 762
13.14.180 10 Bit Programming Register (Lane 6) . . . 763
13.14.181 10 Bit Programming Register (Lane 6) . . . 763
13.14.182 TX ATB Control Register (Set 1) (Lane 6) . . . 764
13.14.183 TX ATB Control Register (Set 2) (Lane 6) . . . 764
13.14.184 TX POWER STATE Control Register (Lane 6) . . . 765
13.14.185 Transmit Control Inputs Status Register (Lane 7) . . . 765
13.14.186 Receiver Control Inputs Status Register (Lane 7) . . . 766
13.14.187 Output Signals Status Register (Lane 7) . . . 766
13.14.188 Transmitter Control Inputs Override Register (Lane 7) . . . 767
13.14.189 Receiver Control Inputs Override Register (Lane 7) . . . 767
13.14.190 Output Signals Override Register (Lane 7) . . . 767
13.14.191 Debug Control Register (Lane 7) . . . 768
13.14.192 Pattern Generator Controls Register (Lane 7) . . . 768
13.14.193 Pattern Matcher Controls Register (Lane 7) . . . 768
13.14.194 Pattern Match Error Counter Register (Lane 7) . . . 769

13.14.195 Current Phase Selector Value. Register (Lane 7) . . . 769
13.14.196 Current Frequency Integrator Value. Register (Lane 7) . . . 770
13.14.197 Scope Control Register (Lane 7) . . . 770
13.14.198 Recovered Domain Receiver Control Register (Lane 7) . . . 770
13.14.199 Receiver Debug Register (Lane 7) . . . 771
13.14.200 RX Control Register (Lane 7) . . . 771
13.14.201 RX ATB Register (Lane 7) . . . 772
13.14.202 8 Bit Programming Register (Lane 7) . . . 772
13.14.203 10 Bit Programming Register (Lane 7) . . . 773
13.14.204 10 Bit Programming Register (Lane 7) . . . 773
13.14.205 TX ATB Control Register (Set 1) (Lane 7) . . . 774
13.14.206 TX ATB Control Register (Set 2) (Lane 7) . . . 774
13.14.207 TX POWER STATE Control Register (Lane 7) . . . 775
13.14.208 PHY Reset Register . . . 775
13.14.209 Transmit Control Inputs Status Register (Broadcast) . . . 776
13.14.210 Receiver Control Inputs Status Register (Broadcast) . . . 776
13.14.211 Output Signals Status Register (Broadcast) . . . 777
13.14.212 Transmitter Control Inputs Override Register (Broadcast) . . . 777
13.14.213 Receiver Control Inputs Override Register (Broadcast) . . . 778
13.14.214 Output Signals Override Register (Broadcast) . . . 778
13.14.215 Debug Control Register (Broadcast) . . . 779
13.14.216 Pattern Generator Controls Register (Broadcast) . . . 779
13.14.217 Pattern Matcher Controls Register (Broadcast) . . . 780
13.14.218 Pattern Match Error Counter Register (Broadcast) . . . 780
13.14.219 Current Phase Selector Value. Register (Broadcast) . . . 780
13.14.220 Current Frequency Integrator Value. Register (Broadcast) . . . 781
13.14.221 Scope Control Register (Broadcast) . . . 781
13.14.222 Recovered Domain Receiver Control Register (Broadcast) . . . 781
13.14.223 Receiver Debug Register (Broadcast) . . . 782
13.14.224 RX Control Register (Broadcast) . . . 782
13.14.225 RX ATB Register (Broadcast) . . . 783
13.14.226 8 Bit Programming Register (Broadcast) . . . 783
13.14.227 10 Bit Programming Register (Broadcast) . . . 784
13.14.228 10 Bit Programming Register (Broadcast) . . . 784
13.14.229 TX ATB Control Register (Set 1) (Broadcast) . . . 785
13.14.230 TX ATB Control Register (Set 2) (Broadcast) . . . 785
13.14.231 TX POWER STATE Control Register (Broadcast) . . . 786
13.15 Transaction, Link, MAC Layers . . . 787
13.16 PCS, PHY Layers . . . 817
13.17 Power Management . . . 817

14 I2C Interface **819**
14.1 Overview . . . 819
14.2 Description . . . 819
14.3 Package Attributes . . . 819
14.4 Registers and Definitions . . . 819
14.4.1 I2C Clock Prescale Register . . . 820
14.4.2 I2C Control Register . . . 821
14.4.3 I2C Data Register . . . 821
14.4.4 I2C Command and Status Register . . . 822
14.4.5 I2C Core Reset Register . . . 823
14.5 Reset . . . 823
14.6 Initialization . . . 824
14.7 Transfer Sequences . . . 824
14.7.1 Example 1: Byte Writes . . . 824
14.7.2 Example 2: Byte Reads . . . 825

14.7.3 Example 3: Unacknowledged Transfer . . . 826
14.8 External Connections . . . 826

15 UART 829
15.1 Overview . . . 829
15.2 Differences, Bugs, and Enhancements . . . 829
15.2.1 Product and Chip Pass Differences . . . 829
15.3 Description . . . 829
15.4 Package Attributes . . . 829
15.5 Registers and Definitions . . . 830
15.5.1 Baud Rate Generation using the Clock Divisor Latch . . . 830
15.5.2 RX/TX Data and Divisor Latch LSB . . . 831
15.5.3 Interrupt Enable Register (IER) and Divisor Latch MSB . . . 833
15.5.4 Interrupt Identification Register (IIR) and FIFO Control Register (FCR) . . . 833
15.5.5 Line Control Register (LCR) . . . 835
15.5.6 Modem Control Register (MCR) . . . 836
15.5.7 Line Status Register (LSR) . . . 837
15.5.8 Modem Status Register (MSR) . . . 838
15.5.9 UART Enable Register . . . 839
15.5.10 UART Reset Register . . . 840
15.6 Reset . . . 840
15.7 Initialization . . . 840
15.8 Interrupts . . . 841
15.9 External Connections . . . 841
15.9.1 Module Service Processor Enabled I/O . . . 841
15.9.2 RS232 Line Voltage Conversion . . . 842

16 Addressing 843
16.1 Overview . . . 843
16.2 Differences, Bugs, and Enhancements . . . 843
16.2.1 Product and Chip Pass Differences . . . 843
16.3 Physical Address Regions . . . 843
16.4 PCI Address Regions . . . 844
16.4.1 Software allocation of PCI address space . . . 844
16.5 General Behavior . . . 845
16.5.1 Access size . . . 845
16.5.2 Read side effects . . . 845
16.5.3 Illegal Addresses . . . 845
16.6 Registers and Definitions . . . 845
16.6.1 Package Attributes . . . 845
16.6.2 Definitions . . . 845
16.6.3 Manufacturer Enumeration . . . 845
16.6.4 Product Enumeration . . . 846
16.6.5 Address Bus Stop Numbers . . . 846
16.6.6 Sub-chip IDs . . . 847
16.6.7 Main Memory Region . . . 849
16.6.8 PCI Memory Region . . . 849
16.6.9 PCI IO Region . . . 849
16.6.10 PCI Config Region . . . 850
16.6.11 Internal SCB Region . . . 850
16.6.12 Internal Non-SCB Region . . . 850

17 Pinout 851
17.1 Overview . . . 851
17.2 Signal List . . . 851
17.3 List of Normal-Mode Signals and Their Test-Mode Overrides . . . 854

18 Programming Considerations **861**
18.1 Overview . . . 861
18.2 Memory Transactions and Ordering . . . 861
18.2.1 The Sync Instruction . . . 861
18.2.2 I-Stream vs. D-Stream Accesses . . . 861
18.2.3 I/O ordering . . . 861
18.2.4 D-Stream vs. I/O Operations and Interrupt Delivery . . . 861
18.2.4.1 I/O read / Block Write interaction . . . 861
18.2.5 Oddball Address Spaces and Physical Addressing . . . 862
18.2.6 Error Traps . . . 862
18.2.7 Interrupts and Interrupt Handling . . . 862
18.2.8 Address Aliasing . . . 862
18.3 The DRAM Controllers . . . 862
18.3.1 Initial Calibration and Setup . . . 862
18.3.2 On-the-fly ReCalibration . . . 862
18.3.2.1 Software filtering of impedance calibration settings . . . 862
18.3.3 DDR Impedance Calibration and Bug 2013 . . . 864
18.4 Initializing the PMI/PCI Controller . . . 864
18.4.1 Unused PCI Controllers . . . 864
18.4.2 PCI Controllers With Connected Devices . . . 864
18.4.3 PCI Controllers With No Connected Device . . . 864

19 Differences, Bugs, and Enhancements **865**
19.1 Overview . . . 865
19.2 User Code . . . 865
19.2.1 Product and Chip Pass Differences . . . 865
19.2.2 Known Bugs and Possible Enhancements . . . 865
19.3 Processor Core . . . 865
19.3.1 Product and Chip Pass Differences . . . 865
19.3.2 Known Bugs and Possible Enhancements (M5KF only) . . . 866
19.4 Addressing . . . 866
19.4.1 Product and Chip Pass Differences . . . 866
19.5 L2 Cache . . . 866
19.5.1 Product and Chip Pass Differences . . . 866
19.5.2 Known Bugs and Possible Enhancements . . . 866
19.6 Memory Controller . . . 866
19.6.1 Product and Chip Pass Differences . . . 866
19.6.2 Known Bugs and Possible Enhancements . . . 867
19.7 PCI . . . 867
19.7.1 Product and Chip Pass Differences . . . 867
19.7.2 Known Bugs and Possible Enhancements . . . 867
19.8 DMA . . . 868
19.8.1 Product and Chip Pass Differences . . . 868
19.8.2 Known Bugs and Possible Enhancements . . . 868
19.9 Fabric Links . . . 868
19.9.1 Product and Chip Pass Differences . . . 868
19.9.2 Known Bugs and Possible Enhancements . . . 868
19.10Fabric Switch . . . 869
19.10.1Product and Chip Pass Differences . . . 869
19.10.2Known Bugs and Possible Enhancements . . . 869
19.11SCB . . . 869
19.11.1Product and Chip Pass Differences . . . 869
19.11.2Known Bugs and Possible Enhancements . . . 869
19.12LBS . . . 870
19.12.1Product and Chip Pass Differences . . . 870
19.12.2Known Bugs and Possible Enhancements . . . 870

19.13 UART 870
19.13.1 Product and Chip Pass Differences 870
19.14 OCLA 870
19.14.1 Product and Chip Pass Differences 870
19.14.2 Known Bugs 871
19.14.3 Possible Enhancements 872

List of Tables

2.1 Timing Budget Spec sheet 53

4.1 DMA Engine Queues 186
4.2 Packet Header and Trailer 200
4.3 Direct Queue Packet Fields 201
4.4 DMA packet fields 201
4.5 DMA End packet fields 202
4.6 Wr_Heap packet fields 202
4.7 Enq_Response Packet fields 202

5.1 **TaskStart Interface from Microengine to Cache Interface** 227
5.2 StartIo Interface from Cache Interface to Microengine 227

6.1 Victimization Rules 295
6.2 Simple L1 Read Miss – L2 Hit 297
6.3 Simple L1 Writeback (All L1 writes hit in L2) 297
6.4 L1 Read Miss, L2 Read Miss, Victim block is in INVALID or SHARE state 298
6.5 L1 Read Miss, L2 Read Miss, Victim block is EXCL, DIRTY, or UPDATED 298
6.6 L1 Read Miss, L2 Read Miss with L1 and L2 evictions 299

7.1 Memory Bus Port Signals From and To Processor Segment X 351
7.2 Memory Bus Port Signals From and To DMA or PCI Segment 352
7.3 Target Addressing 353
7.4 Queue Depth Requirements for CSW Bus Stops 355
7.5 Transfer sequence as a function of address 357
7.6 D-Stream Read to a Non Resident Block: No Victim Writeback 365
7.7 D-Stream Read to a Non Resident Block – With Victim Writeback 366
7.8 D-Stream Read to a Non Resident Block – Hit on Oustanding Read CAM. 367
7.9 D-Stream Read to a Non Resident Block – Hit on Write Back CAM. 368
7.10 D-Stream Read of Cached Data – No Victim Writeback 370
7.11 D-Stream Read of Cached Data – With Victim Writeback 371
7.12 Forwarded D-Stream Read Misses in Probed Cache 372
7.13 D-Stream Read of EXCLUSIVE Block – ORC Hit 373
7.14 D-Stream Read of EXCLUSIVE Block – WBC Hit 374
7.15 I-Stream Read to a Non Resident Block 375
7.16 I-Stream Read to an Non Resident Block: With Victim Writeback 376
7.17 I-Stream Read to a Non Resident Block – Hit on Oustanding Read CAM. 377
7.18 I-Stream Read to a Non Resident Block – Hit on Write Back CAM. 378
7.19 I-Stream Read to a Cached Block in SHARED State 380
7.20 I-Stream Read to a Cached Block: With Victim Writeback 381
7.21 I-Stream Read to a SHARED Block – ORC Hit 382
7.22 I-Stream Read to a Cached Block In EXCLUSIVE State 383
7.23 I-Stream Read to a Cached Block In EXCLUSIVE State: With Victim Writeback 384
7.24 I-Stream Read to a Cached Block In EXCLUSIVE State: With Victim Writeback (Continued from Table 7.23.) 385

7.25 Forwarded I-Stream Read to a Cached Block Misses in Probed Cache . . . 386
7.26 I-Stream Read to a EXCLUSIVE Block – ORC Hit . . . 387
7.27 I-Stream Read to a EXCLUSIVE Block – WBC Hit . . . 388
7.28 D-Stream Read to a Cached Block in SHARED State . . . 390
7.29 D-Stream Read to a Cached Block in SHARED State ORC Hit . . . 391
7.30 D-Stream Read to a Cached Block in SHARED State: With Victim Writeback . . . 392
7.31 D-Stream Write to Invalidate an EXCLUSIVE Dirty Block. . . . 394
7.32 D-Stream Flush to Invalidate and EXCLUSIVE Clean Block. . . . 395
7.33 Block Write to a Non Resident Block . . . 396
7.34 Block Write to a Non Resident Block with a Writeback in Flight from Processor Y . . . 397
7.35 Block Write to a Non Resident Block with a Read in Flight from Processor Y . . . 397
7.36 Block Write to EXCLUSIVE Cached Data . . . 399
7.37 Block Write to EXCLUSIVE Cached Data (continued from Table 7.36.) . . . 399
7.38 Block Write to Cached Data – Collision With Outstanding Write from a Processor . . . 400
7.39 Block Write to Cached Data – Collision With Outstanding Write From a Cacheless Widget . . . 401
7.40 Block Write to Cached Data – Collision With Outstanding Read . . . 402
7.41 Block Write to Cached Data – Encountering an Evicted Block . . . 403
7.42 Block Write to SHARED Data . . . 404
7.43 Block Write to a Cached Block in SHARED State with a Read in Flight from Processor Y . . . 405
7.44 Block Write Collides with Victimization of Target Block . . . 406
7.45 Block Read to Non Resident or SHARED Block . . . 407
7.46 Block Read to Non Resident or SHARED Block – ORC Hit . . . 408
7.47 Block Read to Non Resident or SHARED Block – WBC Hit . . . 409
7.48 Block Read to Cached EXCLUSIVE Block . . . 411
7.49 Block Read to Cached EXCLUSIVE Block – WBC Hit . . . 412
7.50 Block Read to Cached EXCLUSIVE Block – ORC Hit . . . 413
7.51 Block Read to Formerly Cached Block . . . 414
7.52 I/O Register Read . . . 415
7.53 I/O Register Write . . . 416
7.54 Read After Read Hazard ORC Release for RDEX, or RDV following RDEX, or RDV . . . 417
7.55 Read After Read Hazard ORC Release for RDEX, or RDV following RDS, or RDSV . . . 417
7.56 Read After Read Hazard ORC Release for RDS, or RDSV following RDEX, RDV, RDS, or RDSV . 418
7.57 Read After Read Hazard ORC Release for BRD following RDEX, RDV, RDS, or RDSV . . . 418
7.58 Read After Read Hazard ORC Release for RDEX, RDV, RDS, or RDSV following BRD to an UNCACHED Block . . . 418
7.59 Read After Read Hazard ORC Release for RDEX, or RDV following BRD to an EXCLUSIVE Block 418
7.60 Read After Read Hazard ORC Release for RDS, or RDSV following BRD to an EXCLUSIVE Block 419
7.61 Read After Read Hazard ORC Release for RDEX, or RDV following BRD to an SHARED Block . . 419
7.62 Read After Read Hazard ORC Release for RDS, or RDSV following BRD to an SHARED Block . . 419
7.63 Read After Write Hazard WBC Release for BRD, RDEX, RDV, RDS, or RDSV following BWT, WINV, RDV, or RDSV . . . 420
7.64 Interrupt Delivery . . . 422
7.65 Special Commands . . . 423
7.66 CSW Commands, Required Fields . . . 428
7.67 Coherence Controller Command Pipe Actions vs. Tag and CAM Lookups (For transactions that miss in L2 Master Tags) . . . 433
7.68 Coherence Controller Command Pipe Actions vs. Tag and CAM Lookups (For transactions that *hit* in L2 Master Tags in SHARED State.) . . . 434
7.69 Coherence Controller Command Pipe Actions vs. Tag and CAM Lookups (For transactions that *hit* in L2 Master Tags in EXCLUSIVE State.) . . . 435

8.1 Recommended DCLK to CCLK relationships . . . 458
8.2 Supported memory configurations per DDR interface (half of the total main memory connected to each ICE9 chip). . . . 461
8.3 Data Transfer Order . . . 461
8.5 Types of Memory writes: . . . 462

8.7 COH/DDR Interface 463

11.2 LAC Signals 545

12.1 PLL_AB Pins 602
12.2 PLL Bypass Control 603
12.3 PLL VCO Scaling Factors 604
12.4 MuxScan Test Modes 609
12.13Clock Output Control Register (2 copies) 620

15.1 UART Register List 830
15.3 Divisor Latch Values for Common Baud Rates 831
15.8 Interrupt ID Field Definitions 835

Chapter 1

Overview

[Last Modified $Id: chipoverview.lyx 25116 2006-09-07 12:56:52Z wsnyder $]

The SiCortex node chip, ICE9, is the building block for SiCortex dense clusters. Its architecture aims for careful balance between processing power, memory bandwidth, fabric latency, and I/O capability.

1.1 Some History

Way back in January of 2002, Jud Leonard and Matt Reilly got together to figure out what they might be able to do, given they were interested in systems and silicon. So they started talking to people. Lots of different and often strange people. [1]

One of the conversations, with Tom Knight of MIT, turned to high performance technical computing. Tom suggested that what the world needed was a "physics engine" – a device that was specifically designed to solve N-dimensional equations and systems. The traditional supercomputer[2] makers had been in decline for some time. As a result, there wasn't a whole lot of interesting development going on in the field.

Except for clusters.

After that initial conversation, Jud, Matt, and Bryce Denney did a boatload of research.[3] They found that, while the old-fashioned supercomputer and vector computer business had all but died, and traditional symmetric multiprocessors were overpriced and underwhelming in the technical market, the cluster server market was booming. Everywhere they looked, from the Oil Patch (Shell, Exxon/Mobil) to biochemistry, big iron was being replaced by clusters of PC boxes connected with Ethernet or some expensive point-to-point interconnect.

All these machines were being built by the customers. Big iron - like the SGI Origin, the IBM SP2, and the various HP/Compaq machines - was just too expensive. At more than $10,000 per SMP processor node, most customers were abandoning shared memory systems for networked clusters of workstations or 1U rackmount PCs running Linux. The customers had even adopted a common API, called "MPI" (for "Message Passing Interface") as they converted old shared memory codes into message passing applications. But they all ran up against three problems.

First, PC clusters aren't very efficient. We found that the typical application in our target markets would execute about 15 floating point operations (FLOPs) for every access to main memory. So, given a memory access time of about 120nS and a floating point execution rate of, say, infinite FLOPs per second, the average execution rate is just 125MFLOPs. Think about that: customers pay for a widget that runs at 3GHz and can crank out 6 billion floating point operations per second, but they only get 2% of that. All the logic that goes into building a whizzy fast FPU is wasted on these applications. Worse, the 3GHz processor burns about 100W while it spends most of its time waiting on memory.[4]

[1] This is, by no means, an exhaustive history of SiCortex and how we came to be here. The aim here is to outline the thinking and exploring that led to the current architecture. To keep things brief, I've left out the equally important (and far more interesting) story of how we managed to convince four intrepid VC firms to invest in SiCortex.

[2] SiCortex defines a "supercomputer" as a high performance machine that costs more to make than the market will pay. We do not intend to make a "supercomputer."

[3] Google rocks.

[4] Note that we don't believe that the PC designers are misguided. The hell-for-leather strategy that pushed clock rates is a reasonable thing to do for applications that fit in cache. Unfortunately, few technical apps fit in cache – even a very large cache. Desktop apps, however, fit quite nicely. When you consider that $100B is spent on desktop computers every year (compared with $5B or so on technical servers) the big chip guys are probably designing the right widget for their target market.

Second, PC clusters tend to be big. High density clusters might fit 2 Opteron or Xeon processors in a 1U rack slot. But dense packaging produces lots of heat in a small volume. The problem is aggravated by the fact that the building block is a 1U box. A 1U box is just 1.75" high. It is very hard to jam all the parts of a PC in such a box and still have room for airflow. All this conspires to spread a typical 100 node cluster over three or four racks.

Finally, parallel applications on PC clusters are limited by the long latency for message passing. Customers have migrated from shared memory machines to message passing clusters. They developed the MPI specification (it grew out of earlier work on PVM and other message passing schemes) and have implemented it on hardware ranging from simple ethernet controllers to Infiniband, Quadrics, and Myricom hardware. Ethernet based implementations typically impose a cost of 50uS. For about $1,000 per node users could add Infiniband, Quadrics, or Myricom hardware that could get that latency down to abou 5uS. Our models show that the 500nS latency of the SiCortex dense fabric could allow applications to scale to ten or even one hundred times as many processors.

The world probably didn't need a physics engine. But it looked like the world might buy a cluster that was built to run technical applications.

1.2 The System

The SiCortex Dense Cluster is founded upon four piers:

1. Optimize the balance between raw compute rate (FLOPS or Integer Ops/second) and memory latency and bandwidth.

2. Provide low-latency user-mode to user-mode transactions to support MPI.

3. Manage power to provide a high ratio of delivered performance per watt.

4. Aim for an order-of-magnitude advantage in delivered performance per dollar.

This last point is the *raison d'etre*[5] for the SiCortex cluster, so we'd better describe what we mean by "delivered performance." Our model of a technical computation divides the work into three parts: calculation, memory access, and communication. So the time to complete a computation is:

$$T_c = T_{calc} + T_{mem} + T_{comm}$$

Our survey of the applications in the target markets yeilded a large number that, as we said, had a ratio of memory accesses to floating point operations of 1:15. We ignore all the other operations, as most processors will find a way to execute them in parallel (or nearly so) with the floating point ops, or will execute them in parallel with the main memory access. So let's assume that we have an application that needs to do M floating point ops. Then the time to completion is

$$T_c = MT_{FLOP} + \frac{M}{15}T_{access} + T_{comm}$$

For a modern (say 3GHz P4) processor $T_{FLOP} = 0.15nS$ and $T_{access} = 120nS$. Cranking that in to our model:

$$T_c = M\left(0.15 + \frac{120}{15}\right) + T_{comm} \approx M\frac{120}{15} + T_{comm} = 8M + T_{comm}$$

The time to complete the calculation is irrelevant. Applications in this class are all about moving data, and not about doing arithmetic.

That, of course, still leaves the communications (T_{comm}) component. We did a few measurements and found that many applications fell into a range where 1,000 to 100,000 FLOPs were executed for every message sent. So, we cranked in a few numbers. Typical Ethernet based implementations of MPI will consume about 50uS of processor time for every message. Using a rate of say 10K FLOPs per message we get

$$T_c \approx 8M + \frac{M}{10000} \cdot 50000 = (8+5)M$$

Note that $\frac{5}{11}$of the computation is consumed by communication overhead. In actual practice, as the number of processors applied to a problem is increased, the ratio of communication operations to arithmetic operations increases. This is one of the key limiters on parallelism in our target market. As the communication rate approaches

[5] *raison d'etre* (pronounced "rayzohn debtr") French for "reason to be." Often used when an author wants to sound classy.

one message for every thousand FLOPs, the communication overhead begins to dominate the solution time. The SiCortex solution is to reduce the cost of communication to 500nS per operation. This allows practical scaling to many more parallel processes.

Half a microsecond per message is a pretty tough goal. The best-in-class PCI-resident fabric widgets from Myricom or Quadrics get down to 5uS or so. We thought about that problem for a while. What we noticed is that all the previous solutions treated message operations as I/O transactions. They had to: the message widget was out on an I/O bus. But most I/O systems, if they are optimized at all, are optimized for bandwidth, not latency. Good MPI support means providing low latency for short transfers, and high bandwidth for long transfers. Putting an I/O bus (and operating system code, and drivers, and buffer copy operations) between the user's application and the message system puts PC based solutions between a rock and a hard place.

The SiCortex approach is to elevate message operations above the I/O system. By closely coupling the fabric interface to the L2 cache, virtualizing the interface between user mode applications and the fabric, and providing very low latency message routing, the SiCortex system can provide a 10x improvement in message latencies over previous best-in-class approaches. The rest of this document describes the approach in detail.

1.3 ICE9

ICE9 is the central component of a large-scale parallel computer system designed to run technical applications – specifically, those which require large amounts of memory and floating point arithmetic – with superior efficiency. It will run Linux well, and in particular, provide extraordinary performance to MPI, the message passing interface. And we will keep the cost very low.

We will integrate in a single device most of the electronic components needed for the system – microprocessors, caches, memory controllers, fabric switch, DMA engine, and PCI-Express interface. Excluded from the chip are main memory (commodity DRAMs), point-of-load power regulators, and the control/management system.

The fundamental insight behind SiCortex is that faster processor clock speed is no longer an effective way of improving time to solution; that in fact, most parallel technical applications spend the bulk of their time waiting for memory and/or communication between processors.

1.3.1 Goals

Latency We often measure and advertise bandwidth, which sets a strict limit on the throughput available from computer systems, but it's useful to recognize that in most circumstances, latency is the more immediate limitation, because it is generally difficult to get enough parallel activity underway to use the full bandwidth unless each action is brief. This design focusses on main memory (cache miss) latency, which is the primary determinant of single-stream performance in this market, and on MPI communication latency, the time required to get a short message (ping) from a user-mode process on any processor to a waiting user-mode process on the most distant other processor.

- Our goal for the memory latency, measured from a load instruction to use of the data, is 80 ns.
- Our goal for the one-way communication latency, measured by the MPI Ping-Pong test, is 500 ns.
- Our goal for memory bandwidth is 6.4 GBytes/sec, as measured by the McCalpin stream tests.

Power Careful and concerted attention to minimization of power is key to the success of the SiCortex product. By using a small, low-power microprocessor at its most efficient operating point, we are able to keep its cost very low and spread the computational workload over a much larger number of streams. This results in far better utilization of the memory system, which is the bottleneck for delivered performance, but depends on keeping communication delays minimized.

- Our goal for the power dissipation of the chip is 8-10 watts.

Reliability Large-scale systems are particularly sensitive to reliability concerns, for several reasons. On the one hand, the statistical probability of failure is proportional to the number of components, so large systems with many components suffer inherently lower reliability than systems with fewer components. On the other hand, people buy large systems because they have long-running tasks and strong economic incentives to get them finished quickly, so system failures create direct financial consequences for the system owners.

The SiCortex system employs a number of techniques to maximize the reliability of the system from the user's perspective.

- Power consciousness: the system is designed to run cool in the worst case under heavy load, and cooler still when idle, to keep the the reliability high.
- N+1 redundancy of power and cooling systems: Power distribution, from the mains to the module level, is designed with inherent redundancy, so that a failure within the power supply will not cause any interruption of service. Similarly, cooling fans are individually replaceable, and provide enough capacity to maintain specified thermal limits even with one fan inoperative.
- Dual redundancy of the control and management system, allowing the system to survive failures in the control system without effecting normal operation.
- Modular, message-passing hardware/software architecture: failures of a compute node, its memory, or the fabric switches do not force system failures. The fabric architecture is able to route around faults, and the software system is able to restart a checkpointed process on a different processor, so that a failure of one node need not terminate the application(s) using that node.
- Full SEC/DED Error Correcting Code on main memory and L2 cache, to provide fully automatic recovery from transient and permanent single-bit errors as well as early warning of deteriorating devices so that they can be replaced during scheduled maintainance.

1.4 Overall Block Diagram

1.4.1 Processor Cores

ICE9 contains six Mips 5KF processor cores. Each core implements 32KB of instruction cache, 32KB of data cache, and a 256KB “slice” of the shared L2 cache. The L1 data caches, and the shared L2 cache, are coherent.

1.4.2 L2 Cache

ICE9 implements a shared 1.5MB L2 cache. The cache is composed of 256KB slices that are local to a core. The L2 cache controller implements global coherency across ICE9.

1.4.3 Memory Controller

ICE9 implements two DDR2 SDRAM memory controllers. Each controller interfaces to one 72b (ECC) unregistered DDR2 DIMM. This provides for 2GB of memory per node at system FCS, with expansion to 4GB and 8GB as memory technology improves.

1.4.4 PCI-Express Controller

ICE9 implements a PCI-Express controller for I/O. The controller implements 8 PCI-Express lanes, providing 20Gbps of I/O bandwidth per node.

1.4.5 Fabric

ICE9 implements the SiCortex FastFabric, providing three Receive Links and three Transmit Links per node.

1.4.5.1 DMA Engine

The DMA Engine interfaces between the L2 cache and the Fabric switch. It is optimized for MPI operations and allows user applications to send and receive data without invoking the operating system kernel.

1.4.5.2 Fabric Switch

The Fabric switch implements a four-port crosspoint switch among the three fabric links and the DMA engine. The switch provides cut-through routing to minimize latency on packets that are destined for another node. The switch also implements full flow control and error retry to ensure reliable transmission and reception.

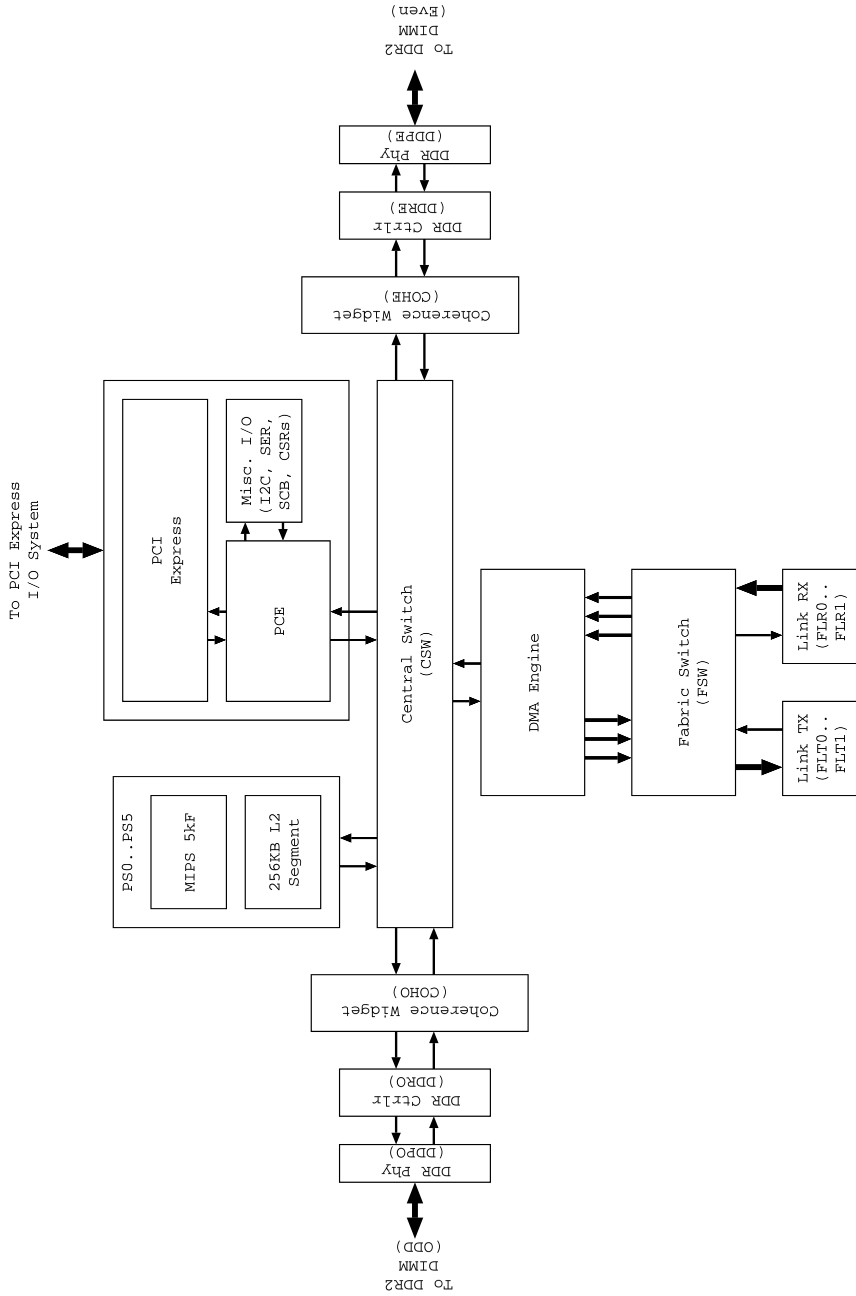


Figure 1.1: The SiCortex Node Chip

1.4.5.3 Link Controllers

Each link controller implement a single FastFabric transmit or receive link.

1.4.5.4 Link Subsystem

The "Link Subsystem" refers to the Fabric Switch, plus the 3 Link Receivers, plus the 3 Link Transmitters. When message is "not for me", it gets sent on to the next ICE9 along the way to it's destination, only passing through the SiCortex FastFabric, and doesn't even enter the DMA Engine in this ICE9. This message will come in a Receive Link, pass through the Fabric Switch, and exit out a Transmit Link. When possible, no "store & forward" occurs, with the Fabric Switch immediately knowing which Transmit Link to use from first-FORD information. If the Transmit Link was available, the beginning of the message is already on the outgoing wires to the next ICE9 before the end of the message has entered this one.

1.4.6 Clock Generator

The clock generator provides internal clocks for ICE9. It generates separate clocks for the cores and L2 caches (nominally 500Mhz, but variable), the PCI-Express controller (always 250Mhz), the memory controller (266Mhz, 333Mhz, or 400Mhz), and the fabric (nominally 200Mhz, but variable).

1.4.7 Miscellaneous

Other on-chip components include the JTAG controller, the on-chip logic analyzer, and on-chip peripherals such as I2C, UART, etc.

1.5 Latency Calculations

1.5.1 Links and Wire-Handling Latency

Latency involved with Link Transmitter and Link Receiver handling sending over a differential pair link between two ICE9 ASICs. The wire propagation delay itself is not included here, but will be included in the table further below.

Unit or Action	Latency	Explanation
Transmit Link unit	4.2 ns	From flopped-in till first bit out on serial line. See Internode Link chapter.
9 more bits onto wire	4.5 ns	Since Transmit Link latency is till first bit out, and Receive Link latency is from when last bit in, we must add this time.
Receive Link unit	15.75 ns	From last bit in on serial line till flopped-out to Fabric Switch. See Internode Link chapter.
receive synchronization	2.25 ns	Receive Link must synchronize incoming 10-bit characters with the local s-clock. This takes 0 to 4.5 ns, depending on phase.
Link Subsystem TOTAL	26.7 ns	

1.5.2 ICE9 to ICE9 Latency

Unit or Action	Latency	Explanation
sending ICE9: software actions	?	
sending ICE9: Processor Hardware	?	
sending ICE9: Central Switch	?	
sending ICE9: DMA Engine	?	
sending ICE9: Fabric Switch	? (>15ns)	From when DMA Engine gives transfer to Fabric Switch, till flopped-in by Transmit Link.
6 hops: "Links and Wire-Handling"	160.2 ns	6 times the 26.7 ns from table above
6 hops: Wire Delay	?	6 times the average wire delay ICE9-ICE9
5 pass-thru ICE9's: Fabric Switch	75.0 ns	5 times minimum Fabric Switch pass-thru latency of 15ns. Defined as from flopped-out by Receive Link till flopped-in by Transmit Link. See Fabric Switch chapter.
receiving ICE9: Fabric Switch	? (>15ns)	From when flopped-out by Receive Link, till when given to DMA Engine.
receiving ICE9: DMA Engine	?	
receiving ICE9: Central Switch	?	
receiving ICE9: Processor Hardware	?	
receiving unit: software actions	?	
6-Hop TOTAL		
5.5 Hop TOTAL		From the 6-Hops total, subtract 20.8 ns and 1/2 of one average wire delay.

1.6 Address Map

All processor cores in an ICE9 see an identical view of the 36 bit physical address space. The address pace is split into three major types of sections: cachable memory space, IO space, and PCI-Express spaces. For more details, see 16.

Chapter 2

Internode Link

[Last Modified $Id: link.lyx 51024 2008-02-15 20:37:33Z rwoodscorwin $]

2.1 Overview

The SiCortex fabric link (we'll call it "the link") is data link with embeded clock, eight-bit wide, differential, all copper, parallel path with a companion serial flow control path. That is, the link is eight lanes of diff pairs, plus one more lane traveling in the opposite direction to carry flow control information. The eight parallel lanes carrying data between nodes is called a "Data Link" or "DL".

Each lane is implemented as a SERDES channel at raw data rate of 2 Gbit/S per lane, or 2 GByte/S per link.

We expect the physical design of the link to be a challenge: some links will traverse only a few inches of PCB trace, while others may travel through several inches of PCB, a connector, up to 30" of backplane, another connector, more backplane, yet another connector, and several more inches of PCB. While daunting, we are encouraged by the fact that several switching systems are carrying significantly higher data rates in similar environments, and that the technology behind channel compensation, reflection cancellation, and low-loss materials have put the SiCortex fabric signalling scheme well within the bounds of current technology.

In order to maintain DC balance on each lane, and in order to detect data-corruption, data traveling on each lane is encoded using a 10B/8B code. Each of the 256 possible 8 bit symbols is recoded into a choice of either of two 10 bit "characters". Out of the 1024 possible 10 bit characters, the encoding only uses those where the number of "1" bits is one greater, one less, or equal to the number of "0" bits. If the number of "1" bits is in excess or deficit, the code is arranged so that at the end of the next symbol transmission the net excess or deficit (over N symbols, for all N) is never greater than 1. That's why there are two 10-bit encodings available for each 8-bit data symbol.

Using 8B/10B encoding scheme, the minimum chunk of data that can be sent over the 8-lane link is 64bits wide every 5nS. We call this chunk a "FORD" (for "Fabric wORD").

The 10B/8B code we have chosen allows for a number of valid 10 bit symbols that have no mapping into the 8 bit space. We use six of these symbols as control and management markers for our link protocols. We use:

K28.0 for ANULL (alternate NULL)

K28.1 for SOLS (start of LinkSync)

K28.2 for EOLS (end of LinkSync)

K28.3 for SOP (start of packet)

K28.4 for EOP (end of packet)

K28.5 for NULL

You may run into the term "ES_COMMA" which means "NULL or ANULL".

We use NULL as an "idle" symbol when the link has no other data to carry, as well as for other purposes. Link will carry data in variable length packets. Each packet begins with an SOP (start of packet) character in lane 0, and ends with an EOP (end of packet) character in lane0.

To keep interfaces clean and to reduce the amount of byte shuffling that goes on in the fabric part of the chip, we'll pass entire FORDs on to the fabric switch logic. The switch datapath is 64bits wide and runs at 1/5 the fabric clock, called the "Switch Clock" or "sclk."

For each 8 bit parallel link from node A to node B, there is a one bit wide serial channel from node B to node A. This link, called the "Control Lane" is used to convey flow-control and buffer status information from a receiving

node back to the node at the other end of the data link. The control lane uses the same 8B/10B dc balance scheme as the data link. As a result, control link tokens are 8 bits wide and arrive every 5nS.

To communicate between two chips, one chip has an FLT (Fabric Link Transmitter) and the other chip has an FLR (Fabric Link Receiver). Between the two chips, Eight data lanes go uni-directionally from the FLT to the FLR, and one control lane goes uni-directionally from the FLR back to the FLT. Inside each chip the FLT or FLR connects to an FSW unit. FSW is described in chapter "The Dense Fabric Switch".

Once a Link has been initialized and is sending traffic, three types of Packets are used. Over the 8-lane-wide data path are sent Data Packets or Idle Packets. Over the one control lane are sent Control Packets. The format of these packets are described near the beginning of chapter "The Dense Fabric Switch", in sections The Data Link and The Control Link.

In these packet formats you will see NULL, ANULL, SOP and EOP, which are recognized by this Internode Link unit for control and management purposes. All other fields within these three packet types will be constructed from the normal 256 8-bit data characters, and are treated as payload by Internode Link and just passed through.

2.2 Differences, Bugs, and Enhancements

2.2.1 Product and Chip Pass Differences

1. NEED IMPL: TWC9A fixes certain noise patterns from causing fabric deadlocks, bug2132.

2. NEED IMPL: All FL internal counters' increment signals should be wired into the SCB counters, bug3488.

2.2.2 Known Bugs and Possible Enhancements

1. Force retraining should always complete, and software shouldn't have to detect and implement retries.

2. The out-of-band path was never used by software, and could be removed for simplicity if desired.

2.3 Reference Documents

AnalogBits QPMA cores are used within the Links to directly drive and receive the differential signals.
AnalogBits documention is checked-in with svn in directory <project>/specs/ice9/AnalogBits/
These are relevant:
ABIPCCE2_datasheet_20051021v2.pdf "ABIPCCE2 Custom PLL DATASHEET".
serdes_PRM_Sicortex_v1_1_2_051130.pdf "Serdes PMA Programmer's Reference Manual".
serdes_test_guidelines_SiCortex_v1_1_1.pdf "Serdes PMA Test Guidelines"

2.4 SERDES Fabric Links

The SiCortex fabric link is eight lane wide in one direction and one lane wide in the other direction. Each lane is implemented as a high speed serial channel at the raw data rate of 2 Gbs per lane. Each lane will use a SERDES transmitter/receiver scheme.

In a SiCortex chassis, fabric links are used for inter-ICE9 data exchange among all 972 ICE9 nodes. Each ICE9 connects to six fabric links, three of those via receive ports while the other three are connected to transmit ports. Each link is a point to point connection between two nodes, so there is a total of (972 x 6)/2 = 2916 fabric links in a chassis.

Each fabric link is built and operates autonomously. The primary function of the fabric link subsystem design is (a) to acquire lane framing on all lanes, (b) to acquire word framing among the eight serial lanes in a data link, (c) to acquire synchronization of the link i.e. bring state of fabric link subsystem to make it usable for data exchange by fabric switch at both ends of link, (d) once link synchronization is acquired then monitor fabric link to detect error conditions and when an error is detected then log the error, (e) after acquiring link synchronization continuously test for loss of link synchronization, and perform re-synchronization of the fabric link when synchronization is lost.

The fabric link subsystem is built using two basic building blocks which are designed by the third party vendor, AnalogBits Inc. They are the lane transmitter which has SERDES PHY, impedance calibration circuitry, PLL, and the lane receiver, which has clock and data recovery circuit. The detailed description of the basic building

blocks is followed by the description of the Fabric Link Transmitter (TxLink or FLT) and the Fabric Link Receiver (RxLink or FLR).

2.5 8B/10B code

The 8B/10B code is implemented as per IEEE 802.3-2002 specifications.

2.6 The Lane Transmitter (Txlane)

A lane transmitter data channel is shown in Figure 2.1. A 10-bit wide data path begins at LaneEncoder. The Txlane latches 10-bit data in aTxDI[9:0] register, serializes it, and transmits serialized bit stream on transmitter PHY. The Txlane transmits LSB (aTxDI[0]) bit first in time and MSB (aTxDI[9]) bit last in time. The data transfer rate is equal in both modules and it is at 10 bits every 5nSec or 10-bits at 200 MHz.

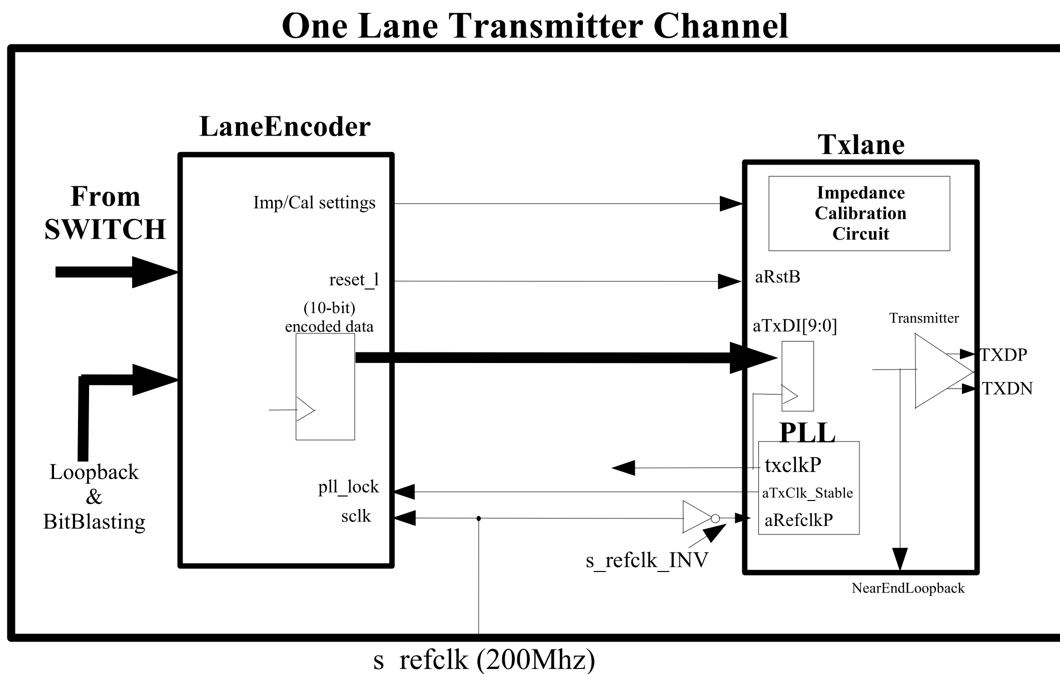


Figure 2.1: Transmitter Lane

The Txlane module has PLL which receives inverted copy of sclk (200 MHz) as refclk and generates txclkP (200 Mhz) in known phase relationship with refclk, which is 2-3 bit period plus propagation delay on internal quad clock tree. The Txlane module uses txclkP as a strobe timing reference signal to transfer data from LaneEncoder. The PLL also asserts a signal, called aTxClk_Stable, indicating when TxClkP is stable and when internal clocks are up and stable.

LaneEncoder operates in sclk domain at 200MHz. LaneEncoder supports 8-bit wide data path from either Fabric Switch or from loopback path within link interface. It has one buffered data stage in sclk domain to perform 8B10B conversion of data. It generates 10-bit wide encoded data every clock tick (at 200 MHz) for Txlane.

The 8B10B tables within TX LaneEncoder have 10-bit busses [9,8,7,6,5,4,3,2,1,0] mapped as [a,b,c,d,e,i,f,g,h,j] on them. TxLane from AnalogBits serialize 10-bit busses [9,8,7,6,5,4,3,2,1,0] such that bit-0 goes first on the serial line, bit-9 last. So, to send bits in the correct order as per IEEE 802.3-2002 spec, LaneEncoder transmits 10-bit bus mapped as [j,h,g,f,i,e,d,c,b,a] to the Txlane.

The data transfer between LaneEncoder and Txlane is synchronous and described in section-2.6.1. The Txlane drives serial data on transmitter PHY at the data rate of 2gbs (giga bits per sec).

The transmitter impedance calibration circuitry controlling Txlane is described in section-2.19.1.

2.6.1 Synchronizer setup between sclk and txclkP

The data transfer between sclk and txclkP is considered synchronous transfer. The synchronous transfer between sclk and txclkP will be achieved by balancing clock layout and placement constraints among synchronizing cells. In each Txlane, there are eleven (11) clock endpoints or targets. Of those 11 endpoints, LaneEncoder has 10 endpoints as clock pins of flops and Txlane has one endpoint as the aRrefclkP input to PLL. The endpoint in TxLane to aRefclkP will be of inverse polarity than to 10 endpoints in LaneEncoder. The design intent is to balance clock tree from common source to 11 endpoints or targets for each Txlane. There are total of 27 Txlanes in ICE9, hence, total of 27 groups of 11 endpoints will be balanced.

The PLL of Txlane generates txclkP which has its rising edge within 2 to 3 bit times (i.e. between 1 nsec-1.5nsec) of serial data rate plus propagation delays on internal clock tree. Design intent is that TxLane will latch data on the rising edge of txclkP.

NOTE : The txclkP clock will not be used by LaneEncoder. For LaneEncoder, txclkP is the implicit clock. However, the goal of synchronous transfer between sclk and txclkP is to meet setup and hold times wrt txclkP in Txlane. .

The synchronous transfer between sclk and txclkP is achieved by allocating timing budget for timing components on clock and data path. The timing diagram of figure-2.2 shows delay component of clock and data path.

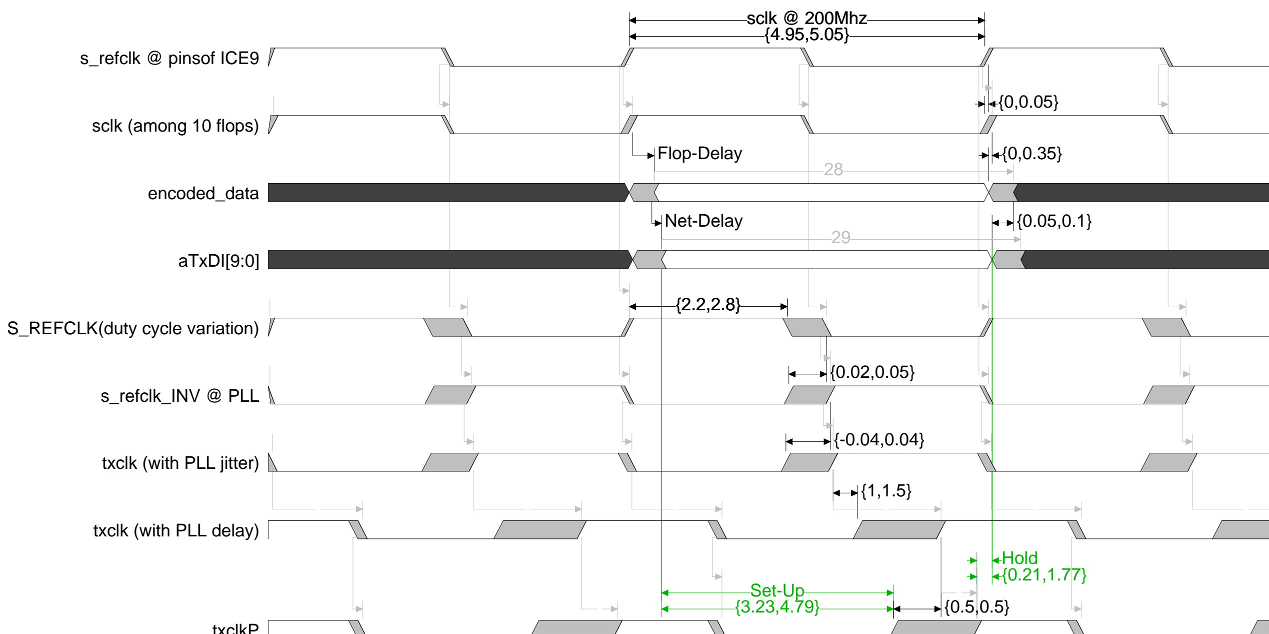


Figure 2.2: synchronizer handshake

NOTE: The clock distribution network in chassis will affect the timing budget of ICE9's internal clock which may not be represented adequately in the timing budget. The s_refclk is signal at the pins

of ICE9. The sclk is clock signal driving 10 target flops in the LaneEncoder. The output of 10 target flops stabilizes at aTxDI[9:0] receiver in Txlane. The data setup and hold checks are to be performed on aTxDI[9:0] wrt txclkP.

A copy of s_refclk is shown with duty cycle variation as S_REFCLK. The inverted copy, called s_refclk_INV, is connected to aRefClk pin of PLL in Txlane. The txclk is the clock output of PLL with PLL jitter spec. The txclkP is the clock output of Txlane wrt to which the setup and hold constraints must be met.

Following spreadsheet specifies timing budget of each component. Note that final design implementation goal is to have setup and hold margin equalize at aTxDI[9:0] cells.

Row		Name	Formula	Min	Max	Margin	Comment
1	V	sclk_period	5	5	5		
2	V	r_jitter	0.05	0.05	0.05		ICE9 clock input spec.= 50ps
3	V	f_jitter	0.05	0.05	0.05		ICE9 clock input spec. = 50ps
4	V	duty_cycle	45	45	45		ICE9 internal spec. = WC 45/55
5	V	duty_cycle_min	((duty_cycle*sclk_period)/100)-(sclk_pe	-0.25	-0.25		
6	V	duty_cycle_max	(((100-duty_cycle)*sclk_period)/100)-(s	0.25	0.25		
7	V	recal_dcycle	[duty_cycle_min,duty_cycle_max]	-0.25	0.25		
8	V	sclk_mismatch	[0,0.05]	0	0.05		ICE9 internal spec. - BC, WC = 0-50ps
9	V	flop_delay	[0.05,0.3]	0.05	0.3		ICE9 internal spec, - BC, WC = [50ps, 300ps]
10	V	delay_line	0	0	0		ICE9 internal spec - delay line (if required), BC, WC = [0, 0]
11	V	rc_delay	[0.05,0.1]	0.05	0.1		ICE9 internal spec - BC, WC = [50ps, 100ps]
12	V	net_delay	delay_line+rc_delay	0.05	0.1		
13	V	INV_mismatch_in_clk	[0.02,0.05]	0.02	0.05		ICE9 internal spec - BC, WC = [20ps, 50ps]
14	P	$p3					
15	V	pll_jitter	[-0.04,0.04]	-0.04	0.04		AnalogBIts PLL spec - WC, short term jitter = 4ps, long term jitter = 40ps
16	V	pll_delay	[1,1.5]	1	1.5		AnanlogBits spec. - BC = 2-bit time, WC = 3-bit time
17	V	clktree_delay	0.5	0.5	0.5		AnalogBits spec. - BC, WC (estimated) = [500ps, 500ps]
18	V	AB_Setup	2	2	2		AnalogBits spec - Setup Constraint of 2ns
19	V	AB_Hold	0	0	0		AnalogBits spec - Hold constrint of 0ns
20	C	Set-Up	[AB_Setup,]	2		<1.23,><1.23,>	<==== Setup Margin
21	C	Hold	[AB_Hold,]	0		<0.21,><0.21,>	<==== Hold Margin
22	P	$p1					

Table 2.1: Timing Budget Spec sheet

2.6.2 Txlane data latency estimates

Data transfer latency estimates are presented below. Latency is calculated from loading of encoded_data to LSB (first) bit on transmitter PHY.

	From	To	BC(ns)	WC(ns)
sclk-to-txclkP	encoded_data	aTxDI[9:0]	3.23	4.79
txclkP-to-????	aTxDI[0]	????	TBD	TBD
????-to-TXD*	????	TXDP/N	TBD	TBD
Total Delay	encoded_data	TXDP/N	xx	xx

2.6.3 Txlane module ports (This port list is not complete. Needs portlist Spec from AnalogBits)

Signal Names	In/Out	From/To	Description
aRefClkP	In	LaneEncoder	Reference clock for PLL at 200 Mhz.
txclkP	Out	–	Transmit Clock signal at 200 Mhz. This signal is not used.
aRstB	In	ICE9 reset distributor	Asynchronous reset signal.
aTxClk_Stable	Out	CSR module	Status signal from PLL indicating the transmit clock is stable.
aTxDI[9:0]	In	LinkEncoder	10-bit data which is 8B10B encoded. Txlane accepts this data at frequency of sclk (200 MHz).
TXDP/TXDN	Out	Primary output pins	Differential PHY output. Txlane will drive this data at 2 gbs.

2.6.4 8B10B code Validation Plan

Verification team will get 8b10b code standard from IEEE 802.3 (ethernet) spec. Verification team will verify and validate each Txlane against 802.3 spec, including negative cases of errors.

2.6.5 Verification Checklist: (This section is not complete)

1. Verify sclk/txClkP synchronizer settings

2. Verify reset function

3. Verify driving bad disparity function tx

4. Verify driving invalid character on tx

5. Verify NearEndLoopback mode

2.7 The Lane Receiver (Rxlane)

A lane receiver data channel is shown in Figure-2.3. The Rxlane module of the diagram will be delivered by AnalogBits, Inc.

The receiver datapath begins in Rxlane at the differential inputs, Rxdp and Rxdn, of the SERDES receiver PHY. The baud rate at Rxdp/Rxdn is 2Gbps. The embeded data and clock signals from PHY are separated by the Rxlane. The Rxlane de-serializes incoming data stream, and drives databus aRxDO[19:0] and clock aRxClkN in the source synchronous mode to the Framer module. The aRxClkN signal is extracted clock from incoming data stream and it is operating at 200 MHz. The content of aRxDO[19:0] has data fields in the form of <current_10bits,previous_10bits>. The Rxlane transmits MSB (aRxDO[19]) bit which has the most recent bit arrived on PHY and LSB (aR<xDO[0]) bit which has the earliest arrived bit on PHY.

The 8B10B tables within Framer has 10-bit busses [9,8,7,6,5,4,3,2,1,0] mapped as [a,b,c,d,e,i,f,g,h,j] on them. The RxLane from AnalogBits de-serialize 10-bit bus [j,h,g,f,i,e,d,c,b,a] such that bit-a is received first from the serial line, bit-j last and drives it in that order to Rxlane. So, to receive bits in the correct order as per IEEE 802.3 spec, Framer maps received 10-bit bus as [a,b,c,d,e,i,f,g,h,j].

The data transfer rate in both modules, Rxlane and Framer, is equal and it is at 10-bits every 5 nSec or 10-bits at 200 MHz.

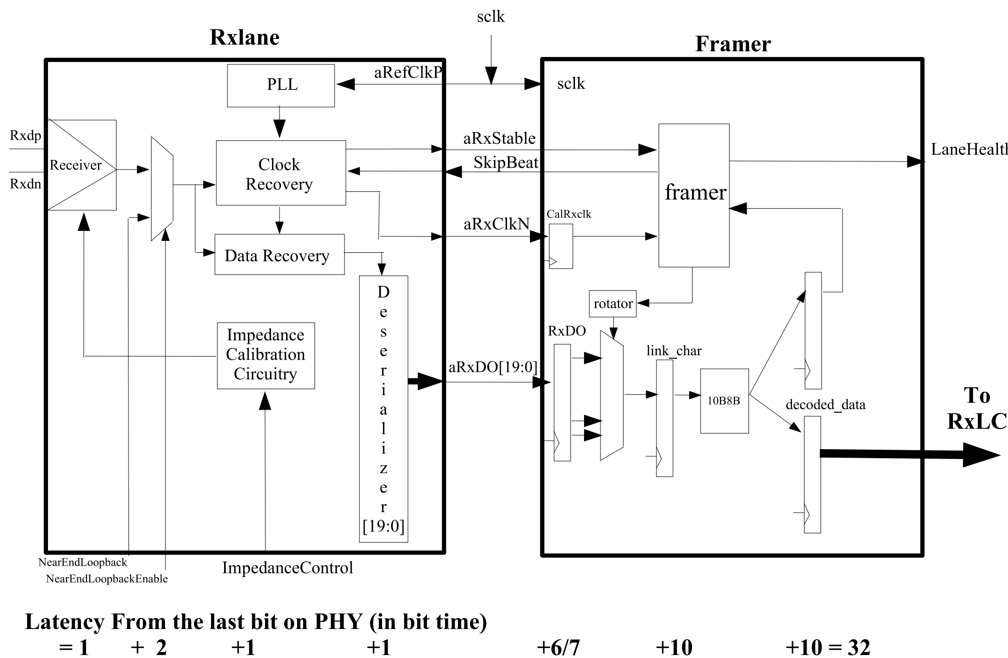


Figure 2.3: Receiver Lane

The Rxlane has PLL which gets a copy of sclk (200 MHz) on its aRefClkP pin. The SiCortex system uses a single oscillator to drive the primary clock distribution tree. A copy of the primary clock distribution tree is referenced by the transmitter to drive the transmitter PHY. Because the origin of both clocks, aRefClkP and the transmitter clock, is the same oscillator, the difference in frequency between aRefClkP and recovered clock, aRxClkN, from the

receiver PHY is 0-ppm. It is important to note that the received clock aRxClkN is extracted from the incoming datastream, and not from the aRefClkP, even if it does connect to aRefClkP prior to starting CDR (clock and data recovery) function.

The detailed description of the Framer module is described in section-2.7.2.

2.7.1 Clock Alignment and Synchronizer setup between Rxlane and Framer transfer

The clock alignment between aRxClkN and sclk must take place after both clocks, aRxClkN and sclk, are stable, PLLs are locked, and the reset signal is deasserted. All data trasfers between Rxlane and Framer are ignored before the clock alignment step is complete.

Rxlane and Framer handshake exploits the fact that the aRxClkN and aRxDO[19:0] is the source synchronous transfer. Please note following four salient points about data tranfer between Rxlane and Framer module.

1. The Framer logic design will sample state of aRxClkN signal to find alignment between two clocks, aRxClkN and sclk, and then adjust aRxClkN for data synchronizing transfer between Rxlane and Framer.
2. The Framer logic design will not use aRxClkN clock to strobe data transfer from aRxDO[19:0].
3. AnalogBits design team will be matching electrical delays on 21 signals, aRxClkN and aRxDO[19:0], from internal cells of IP to output port of Rxlane.
4. The Sicortex design team will be matching electrical delays on 21 signals, aRxClkN and aRxDO[19:0], from port of Rxlane to receiver cells in Framer.

The frequency of sclk and aRxClkN is identical and it is 200 Mhz. However, the phase relationship of aRxClkN wrt sclk is in-determinate because the phase relationship between the two clocks depend on the electrical length of the receiver lane. For aligning aRxClkN with sclk, Rxlane will allow shifting the phase of aRxClkN in increments of 1-bit time. The Rxlane will shift the phase of aRxClkN by stretching aRxClkN clock by 1-bit time. The clock stretching will not be a glitchless operation, however, sampling of aRxClkN will be performed only after the clock alignment operation is completed.

2.7.1.1 SkipBeat Handshake

Refer to Section-8.1 of "Serdes PMA Programmer's Reference Manual" for the details of the SkipBeat handshake. The SkipBeat timing parameter table for Sicortex design is shown below:

Parameter	Units	min	max	typ
Nskipbeaton	aRxClkN period	3 - (15ns @ 200 Mhz)	3 - (15ns @ 200 Mhz)	3 - (15ns @ 200 Mhz)
Nskipbeatrepeat	aRxClkN period	31 - (155ns @ 200 Mhz)	31 - (155ns @ 200 Mhz)	31 - (155ns @ 200 Mhz)
Tskipeffective	ns	TBD	TBD	TBD

The algorithm for aligning aRxClkN with sclk is described below:

```
Begin:
   First_Search :
      Move phase of aRxClkN by 1-bit time.
      Test logic level of aRxClkN.
      If it is 0 then set flag-First_Search and jump to Second_Search else repeat.
   Second_Search :
      Move phase of aRxClkN by 1-bit time.
      Test logic level of aRxClkN.
      If it is 1 then set flag-Second_Search and jump to Final_Search else repeat.
   Final_Search :
      Move phase of aRxClkN by 1-bit time.
      Test logic level of aRxClkN.
      If it is 0 then set flag-Final_Search and jump to Adjustment else repeat.
   Adjustment:
     Move phase of aRxClkN by 5-bit times, set flag-Adjustment and exit.
End:
```

After the receiver clock, aRxClkN, is stable, the worst-case time to complete the skipbeat handshake at 200 Mhz is calculated as below:

1. The maximum time taken in First_Search = 5 skipbeat operations x 31 sclk periods = 5 x (31 x 5) = 775ns
2. The time taken in Second_Search = 5 skipbeat operations x 31 sclk_periods = 5 x (31 x 5) = 775ns
3. The time taken in Final_Step = 5 skipbeat operations x 31 sclk_periods = 5 x (31 x 5) = 775ns
4. The time taken for adjustment = 5 skipbeat operations x 31 sclk_periods = 5 x (31 x 5) = 775ns
5. Total time in skipbeat handshake = 1 + 2 + 3 + 4 = 3100ns

2.7.1.2 The RxClk alignment

The clock alignment and receiver synchronizer timing diagram is shown in Figure-2.4.

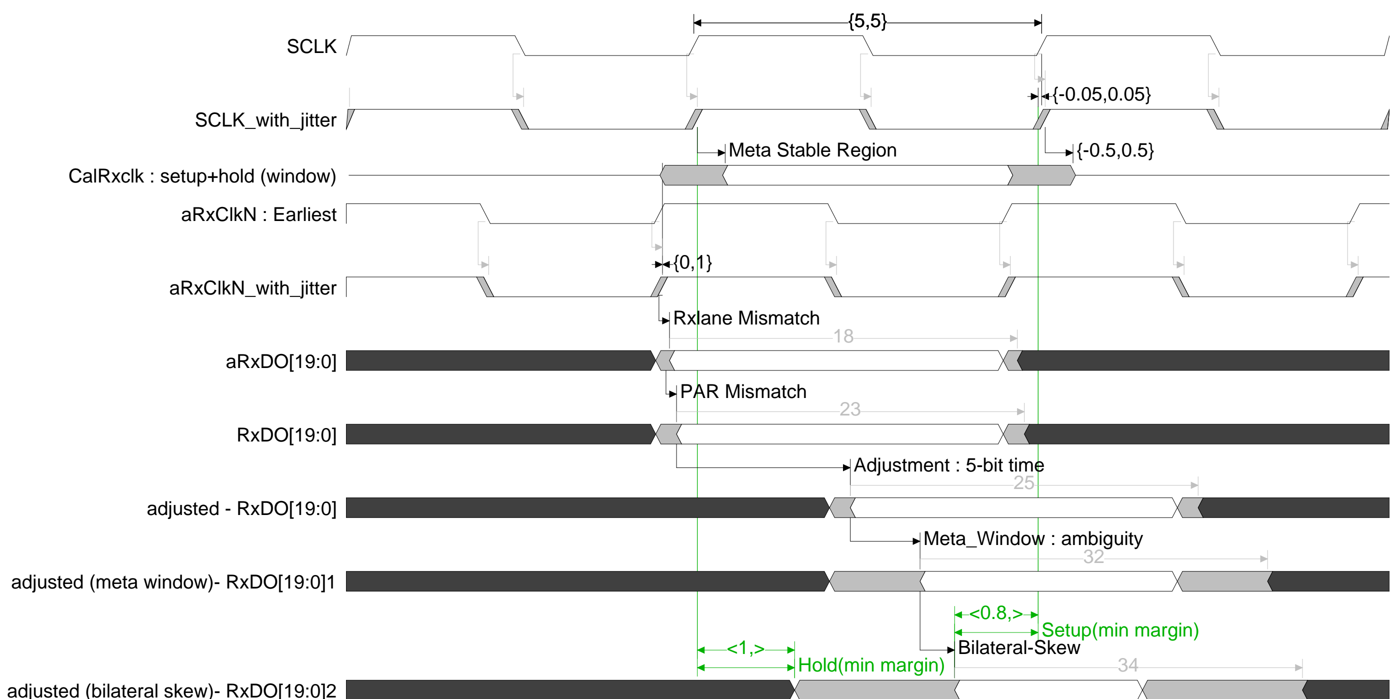


Figure 2.4: Clock alignment and Receiver Synchronizer

The timing diagram shows that sclk at 200 Mhz. After adjusting for the jitter spec of sclk, CalRxclk signal shows metastability region of a flop in TSMC-90G. The timing diagram shows early arrival of aRxClkN signal transitioning from low-to-high (which will be sampled as going from high-to-low! This is non-intuitive but sampling state of CalRxclk will observe its output state going from 1 to 0). After accounting for datapath and clock path mismatch, the databus at RxDO[19:0] will have valid data window which is equal to period of sclk minus data path mismatch between Rxlane and Framer. The final adjustment of 5-bit time for aRxClkN provides equalized set-up and hold time at RxDO[19:0] register.

The timing budget for each timing component in clock alignment data path is shown in Figure-2.5. The last column which is a comment column shows ownership of each line item. The “ICE9 spec” items are owned by Sicortex while “AnalogBits Spec” items are owned by AnalogBits.

2.7.2 The Framer Module

The Framer module interfaces with Rxlane in slow clock (sclk at 200 Mhz) domain. The block diagram of the Framer module is shown in Figure-2.3. There are two primary tasks of Framer module are described below.

Row		Name	Formula	Min	Max	Margin	Comment
1	V	sclk_period	5	5	5		ICE9 clock spec : 200 Mhz
2	V	sclk_rise_jitter	[-0.05,0.05]	-0.05	0.05		ICE9 clock input spec. BC, WC = -50ps, 50ps
3	V	sclk_fall_jitter	[-0.05,0.05]	-0.05	0.05		ICE9 clock input spec. BC, WC = -50ps, 50ps
4	V	setup_plus_hold	[-0.4,0.4]	-0.4	0.4		TSMC 90g FLOP spec : exaggerated window
5	V	early_valid_clk	-0.5	-0.5	-0.5		sclk_rise_jitter(min) + setup_plus_hold(min) - CDR_rise_jitter(max)
6	V	CDR_rise_jitter	[-0.05,0.05]	-0.05	0.05		AnalogBits PLL spec. - BC, WC = -50ps, 50ps
7	V	CDR_fall_jitter	[-0.05,0.05]	-0.05	0.05		AnalogBits PLL spec. - BC, WC = -50ps, 50ps
8	V	Rxlane_mismatch	[0,0.1]	0	0.1		AnalogBits spec - BC, WC = 0-100ps
9	V	PAR_mismatch	[0,0.1]	0	0.1		ICE9 spec. - BC, WC = 0-100ps
10	V	adjust_bit_time	5	5	5		ICE9/AnalogBits spec. = Synchronizer Adjustment in bit-time
11	V	meta_window	[0,sclk_period/5]	0	1		ICE9 spec : Metastable window of a flop = 1ns
12	V	bilateral_skew	[-0.5,0.5]	-0.5	0.5		AnalogBits spec in bit-time - BC, WC = -500ps, 500ps
13	V	setup_check	0.4	0.4	0.4		TSMC 90g Flop setup check
14	V	hold_check	0.4	0.4	0.4		TSMC 90g Flop hold check
15	C	Setup(min margin)	[setup_check,]	0.4		<0.8,><0.8,:	TSMC 90G, setup constraint check
16	C	Hold(min margin)	[hold_check,]	0.4		<1,><1,>	TSMC 90G, hold constraint check

Figure 2.5: Clock alignment timing budget

2.7.2.1 The clock alignment and synchronizer setup

The clock alignment and synchronizer setup with Rxlane is described in section-2.7.1.

2.7.2.2 Framing Function and flag-LaneHealth

The Rxlane receives serial data stream without any indication of framing boundary and passes 20 bits of de-serialized data, aRxDO[19:0], at 200 Mhz to Framer. The content of aRxDO[19:0] has data fields in the form of <current_10bits,previous_10bits>. The Rxlane transmits MSB (aRxDO[19]) bit which has the most recent bit arrived on PHY and LSB (aTxDO[0]) bit which has the earliest arrived bit on PHY. The Framer has to find framing boundary of incoming data stream. To aid framing function, all lane transmitters in ICE9 will drive k_28.5 while framing function is active.

The data on serial lane is 10-bit encoded data, so there are 10 possible framing boundaries within incoming serial data. Only 1 of those 10 framing boundaries is a valid framing boundary. The Framer forms 10 possible character strings of incoming data stream. It is assumed that each string is given an identifier, starting from 0 to 9.

A framing controller, called framer, has 10-stage counter called rotator. Rotator stages are from 0 to 9. The rotator stage is used to select character string identifier.

Framer will find framing boundary of incoming data stream by setting rotator to a stage for 64 consecutive clock cycles. Framer will validate framing boundary, if and only if, it has received valid K28.5 (NULL) characters without disparity errors for at least 48 cycles. If framer has not found framing boundary then then it will increment rotator stage and perform above test again. There are only 10 possible framing boundaries in free running data stream, hence above scheme will find framing boundary in about (64 x10) = 640 characters.

Time to send 1 character on link is 5nS, so framing will take about (640 x 5) = 3.2uSec.

When Framer is successful in finding a frame within incoming data stream, it sets the flag-LaneHealth indicating that it is receiving error free K28.5 characters from Rxlane and lane's health is declared "good".

After setting flag-LaneHealth, framer switches its function to check for the condition of loss of framing using credit based algorithm. The framer may now receive ANY of the data or control characters. The framer assigns health_rating of 0xF to the lane. A lane can not receive higher than 0xF count of health_rating and lane can not receive lower than 0x0 count of health_rating. When health_rating of a lane reaches 0x0, lane is non-usable and it is declared "bad" and indicated so by clearing of flag-LaneHealth.

The Framer receives a character from a serial lane at the rate of 200 Mhz. The Framer evaluates every character it received and determines if it is a credit or a debit. A character without an error is a credit and a character with an error is a debit. The framer adjusts lane's health_rating for every character. If health_rating of lane ever reaches 0x0 then framer determines that lane has lost framing, its health status is bad, and clears flag-LaneHealth. When flag-LaneHealth is reset, the framer re-enters the framing function.

2.7.3 The Wordsync function

The fabric switch transmits and receives 64-bit data (or FORD) to/from the link. Though the transmit link transmits data on eight transmit lanes wrt to sclk, due to eight seperate physical paths taken from one ICE9 to

another, the data propagation delay mismatch may result among eight lanes. In such case, the Wordsync module in receiver may observe mismatched arrival times on eight data lanes. The Wordsync function equalizes electrical delays among eight receiver lanes. The Figure-2.6 shows implementation details of the Wordsync function.

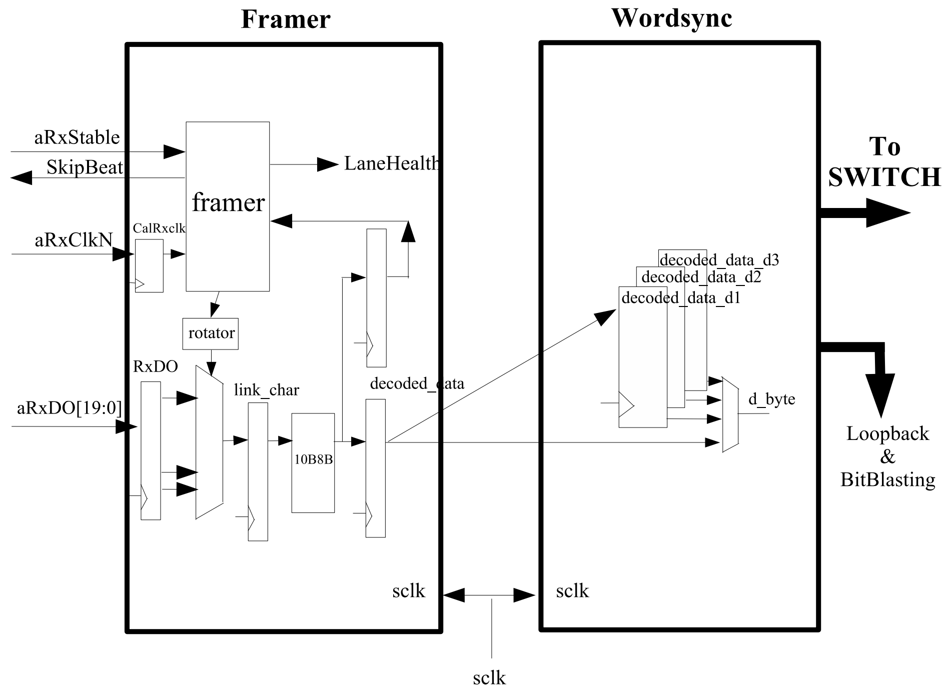


Figure 2.6: Wordsync Function

The Wordsyncing among 8 receiver lanes is achieved in three steps. First step is to measure the propagtion delay differences among eight lanes. The next step is to increase the electrical delays of the faster lanes (and thus making them slower). Final step is the validation step of verifying that the total propagation delay of eight receiver lanes is equal.

The Wordsync module has provision to delay data byte received from the receiver lane by either 1 or 2 or 3 sclk periods.

To measure the propagation delay difference among eight receiver lanes, a special character k28.1, is sent by the transmit link on 8 transmitter lanes on the same rising edge of sclk and then in eight Wordsync modules of the receiver link, the arrival time of k28.1 are is noted. The lanes receiving k28.1 earlier are faster. The Wordsync module can measure propagation delay difference of upto 3 sclk periods among eight receiver lanes. In next step, the Wordsync module will increase data propagation delay of the faster lanes and make them equal on all 8 receiver lanes. The final step is the verification step. In this step, the transmitter will transmit a special character k28.1 again and receiver lanes will validate that all eight of them received k28.1 in the same sclk cycle. Next, the transmitter transmits all 534 valid 8B10B characters, each character twice, on all 8 lanes. Upon receiving all (536 x 2) characters without an error on all receiver lanes completes the wordsync function.

2.7.4 Rxlane to Framer data latency estimates

Data transfer latency estimates is presented below. Latency is calculated from the last bit (MSB bit of aRxDO[19:0]) on receiver PHY to decoded_data in Framer module in bit time.

	From	To	bit-time
PHY delay	Rxlane	Rxlane	1
Rxlane multiplexer	Rxlane	Rxlane	2
CDR	Rxlane	Rxlane	1
Load Deserializer	Rxlane	Rxlane	1
aRxDO[19:0]	Rxlane	Framer	7
link_char	Framer	Framer	10
decoded_data	Framer	Framer	10
Total Delay	Rxdp/Rxdn	decoded_data	32

2.7.5 Rxlane module ports

Signal Names	In/Out	From/To	Description
aRxClkP	In	Framer	Reference clock signal for PLL.
aRxStable	Out	Framer	Status signal indicating that extracted clock aRxCLKN is stable.
SkipBeat	In	Framer	Handshake signal for clock alignment between Framer and Rxlane. When asserted, Rxlane will skip aRxClkN clock by 1-bit time.
aRxDO[19:0]	Out	Framer	Deserialized 20-bit data from receiver PHY. The content of aRxDO[19:0] has data fields in the form of <current_10bits,previous_10bits>. The Rxlane transmits MSB (aRxDO[19]) bit which has the most recent bit arrived on PHY and LSB (aTxDO[0]) bit which has the earliest arrived bit on PHY. This databus is source synchronous to aRxClkN at 200 Mhz.

2.7.6 8B10B code Validation Plan

Verification team will get 8b10b code standard from IEEE 802.3 (ethernet) spec. Verification team will verify and validate each Rxlane against 802.3 spec, including negative cases of errors.

2.7.7 Verification Checklist:

1. Verify aRxClkN/sclk synchronizers
2. mis-alignment of aRxClkN among group of 8
3. verify SkipBeat function
4. verify Skipbeat offset variable
5. verify manual operation of clock alignment (or SkipBeat function)
6. Verify force lane-health function
7. Verify enable/disable lane health
8. Verify force Wordsync function
9. Verify Wordsync function through SCB
10. Verify number of data pattern selection in Wordsync function

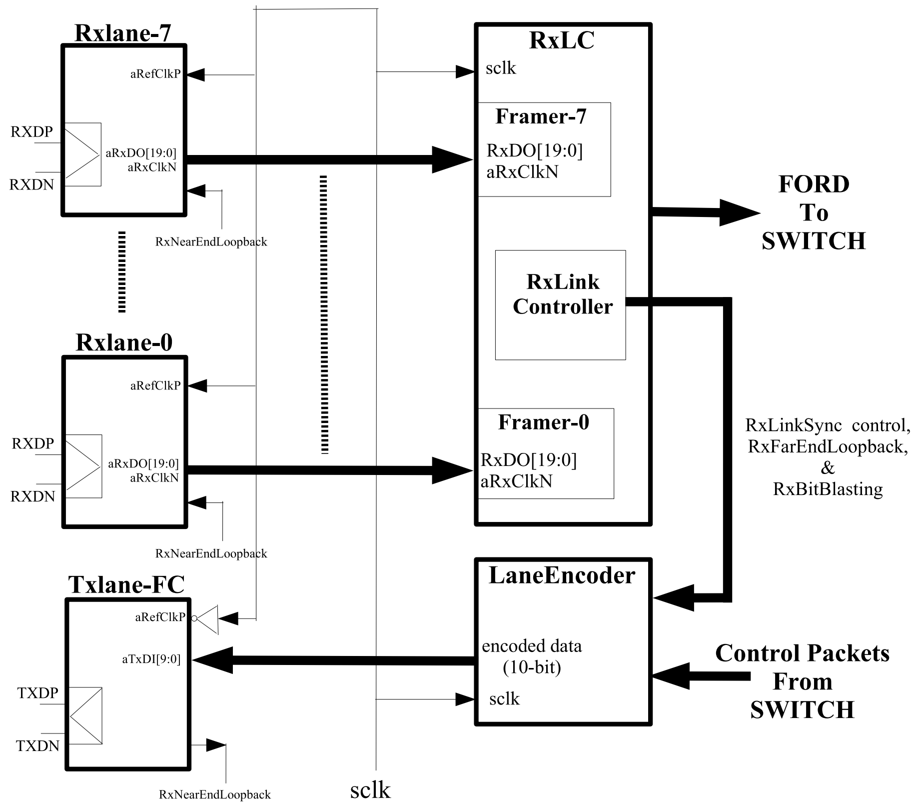


Figure 2.7: Receiver Link

2.8 The Fabric Link Receiver

The Fabric Link Receiver (RxLink or FLR) has eight serial receiver lanes for the data packet transfers and one serial transmitter lane for the control packet transfers. The RxLink is constructed from eight Rxlane modules, one TxLane module, one RxLC module which contains eight Wordsync modules, and one LaneEncoder module as shown in Figure-2.7.

The Fabric switch clock, sclk @ 200 MHz, is distributed to the RxLC and LaneEncoder modules as its primary clock. Copy of sclk is also distributed to the Rxlane-7 through 0 on their aRefClkP pins. Inverted copy of sclk is distributed to the Txlane-FC on its aRefClkP pin.

Eight Rxlanes, numbered 7 through 0, receive serial data on receiver PHY, receover data and clock from the incoming serial data stream, and drive deserialized data aRxDO[19:0] and clock aRxClkN to eight Wordsync modules. Eight Wordsync modules, numbered 7 through 0, are within the RxLC module. Each Wordsync module handshakes with one Rxlane module to setup the synchronizer transfer from the Rxlane and then acquire framing from incoming data stream. Then the RxLC module acquires the Wordsynchronizations among eight Wordsync modules. After acquiring Wordsynchronization, the RxLC module decodes received data from Rxlanes and then transfers 64-bit FORD and 3-bit status (SOP, EOP, IDLE) to the fabric Switch every sclk cycle.

The control packets travel in the opposite direction. The control packets originate in the fabric switch. From there, they travel through the LaneEncoder where they get 8B10B encoded. Encoded data from LaneEncoder move to aRxDI[9:0] register of the Txlane-FC, which serializes and transmits data on transmitter PHY.

The RxLC module has another output port for supporting RxLink bringup routine, FarEndLoopback and BitBlasting modes.

The RxLC has a controller called RxLink Controller (RxLC) which has 2 main functions.

1. Acquire link synchronization by executing the hardware routine called RxLinkSync. Upon successful completion of RxLinkSync, RxLink has acquired link synchronization.

2. After successful completion of RxLinkSync, RxLC enters the state of MissionMode during which RxLink is functional and the fabric switch at both ends of the RxLink can exchange data and control packets. In MissionMode, RxLC will act as a link supervisor and keep checking for link errors including loss of link synchronization. When it detects that link synchronization is lost, then RxLC exits MissionMode and enters hardware routine RxLinkSync for re-synchronization of RxLink.

In hardware execution routine RxLinkSync, RxLC controller communicates with hardware execution routine (called TxLC and described later in section-2.9) of corresponding 8-lane transmitter of ICE9 using return path through LaneEncoder. In hardware execution routine, RxLC is the master and TxLC is the slave. The RxLinkSync routine gets executed once after power is up, and after PLLs are locked, and the reset signal is negated. The RxLinkSync routine is entered from MissionMode if loss of link synchronization is detected.

Loss of synchronization, i.e. loss of heartbeat, will occur when any of the following conditions is detected. Clearing of flag-LaneHealthStatus due to any of the following cases.

(a) loss of signal on serial receiver or due to excessive character errors and/or disparity errors on any of the 8 data lanes,

(b) setting of flag-ForceRetraining through SCB (see section-2.8.1 for explanation),

(c) clearing of flag-Heartbeat from heartbeat timeout on data-lane-0 (see section-2.8.1),

(d) disabling RxLink with SCB RxLcControl Ena bit, SoftReset, or hard reset line.

2.8.1 Status Flags required by RxLinkSync and RxLC

RxLC will have following status flags.

1. Flag-AllRxlanesReset The flag-AllrxlanesReset is set when PLL of all eight RxLanes are locked, and reset signal is deasserted in all eight RxLanes in their rxfclk domain, and reset signal in sclk domain is deasserted. This flag is reset when any one of eight PLLs of RxLanes has lost lock, or any one of eight RxLanes has reset signal asserted, or reset signal in sclk domain is asserted.

2. Flag-LinkHealth Each RxLane provides status of lane's health in real time through a flag-LaneHealthStatus. A flag-LaneHealthStatus is set if RxLane has acquired frame, and it is receiving valid data and control characters from receivers without disparity errors, otherwise flag is reset. Software may also reset flag-LaneHealthStatus through SCB by setting ClrLaneHealth. There are 8 flags, flag-LaneHealthStatus. The flag-LinkHealth is created from lane health status. The Flag-LinkHealth is set if all 8 flag-LaneHealthStatus are set otherwise it is clear.

3. Flag-ForceRetraining The hardware execution routine RxLyncSync may be initiated through SCB by setting flag-ForceRetraining. The transition from 0 to 1 of flag-ForceRetraining causes RxLyncSync to be initiated. Software should then clear flag-ForceRetraining so it is available for future use.

4. Flag-Heartbeat During MissionMode the RxLink uses a "heartbeat" method of detecting good communication from the TxLink in the other chip. The following steps describe heartbeat operation:

- When the RxLink achieves MissionMode, flag-Heartbeat is set.
- The fabric switch drives data packets using 8 lanes. The data packets are bounded by SOP (start of packet char, k28.3) and EOP (end of packet char, k28.4) characters. The transition from SOP and EOP characters to non-SOP and non-EOP characters detects the heartbeat.
- When the fabric switch is idle, it drives IDLE packets. The transmitter link will drive IDLE packets on lane-0 using NULL (k28.5) or AIDLE (alternate idle char k28.0) characters on link. During idle cycles, transition from AIDLE character to non-AIDLE character detects the heartbeat.
- During MissionMode, if heartbeat is not detected for consecutive 128 clock cycles, by either of the above methods, then it is assumed that link has lost heartbeat and flag-Heartbeat is cleared, otherwise it remains set.
- A loss of Heartbeat causes a loss of MissionMode, and routine RxLyncSync is re-entered.
- Also, if MissionMode is lost for any other reason, flag-Heartbeat will be cleared.

5. Flag-RxLinkSync The flag-RxLinkSync is a status flag. It is controlled by Wordsync module. This flag is set when link is executing hardware routine RxLinkSync, otherwise this flag is clear. This flag will remain set if hardware routine RxLinkSync has encountered a failure and/or hardware routine RxLinkSync has not been completed successfully.

6. Flag-MissionMode The flag-MissionMode is a status flag. It is controlled by Wordsync module. This flag is set when hardware execution routine RxLinkSync has been successfully completed i.e. routine has been successful in acquiring lane framing, and word framing. When flag-MissionMode is set, it indicates that RxLink is operational and control of a link has been transferred to the fabric switch. Setting of flag-MissionMode implies that (i) fabric switch at both ends will maintain link heartbeat on data transfer in both direction, (ii) spurious lane errors will be detected by lane controllers as bit errors, and those errors are logged, (iii) spurious bit errors will not make link unusable, and (iv) persistent bit errors on one or more lanes will cause loss of link health by resetting one or more flag-LaneHealth(s), which in turn, will force re-entry of the hardware execution routine RxLinkSync by resetting flag-MissionMode and setting flag-RxLinkSync.

2.8.2 RxLinkSync Routine

Jump to Begin:

- BEGIN:
  If (flag-ForceRetraining) then jump to Step-1

- Step-1:
  Set flag-RxLinkSync, reset flag-MissionMode, reset flag-Heartbeat.
  Force Idle on FORD-to-FabricSwitch, Disable data path from ControlPacket-to-link, Force k_28.5 on TxLane (send NULL)
  (sending k_28.5 without Heartbeat packet will force TxLC to jump to TxLinkSync Routine)
  Wait till flag-LinkHealth is set, then jump to Step-2.
  (when flag-LinkHealth is set, then all lanes are receiving k_28.5)
  Note: If flag-LinkHealth is asserted for less than 3-ticks, then controller will not jump to Step-2 and will re-enter or remain in Step-1. For each occurance of such case, or each jump to Step-2, R_FlrxRxLcCount will be incremented.

- Step-2:
  If (~flag-LinkHealth) then jump to Step-1
  else
  Force k_28.5 on TxLane (send NULL)
  Wait for time T1 = (R_FlrxRxLcControl.Step2WaitTime number of sclks), where
  T1(min) = (Rate of Heartbeat + 4 times maximum link delays) = (100 * sclk period) + 4 * 10nS = 500 + 40 = 540nsec
  (sending k_28.5 without Heartbeat packet will force TxLC to jump to TxLinkSync Routine)
  Jump to Step-3

- Step-3:
  If (~flag-LinkHealth) then jump to Step-1
  else
  Force pattern of k_28.5 and k_28.0 (send alternate NULL characters at the rate of once every 256 sclk cycles)
  Wait to receive pattern of k_28.5 and k_28.0 (wait for TxLinkSync routine to respond)
  Jump to Step-4

- Step-4:
  If (~flag-LinkHealth) then jump to Step-1
  else
  (Note: This is the Wordsync Routine. Refer to Section-2.7.3 for details of the operation.)
  Send and wait for first request to return SOLS char (delay calibration cycle request to return SOLS character k28.1)
  Send and wait for second request to return SOLS char (word alignment cycle request to return SOLS character k28.1)

Send and wait for valid verification data patterns (512) and control chars (24) , EOLS (k28.2) being the last one
After that, send NULLs (k28.5), while still in Step-4.
if (Wordsync error, or data verification error) then
Set error bits and stay in Step-4 until ˜flag-LinkHealth.
else
Wait for EOLS to come back from TxLink in other chip (with no time limit) and then
Set flag-Heartbeat, and jump to END

- END: (enter MissionMode operation)
  Set flag-MissionMode, reset flag-RxLinkSync.
  Enable data packet path from RxLink-to-FabricSwitch.
  Enable control packet path from FabricSwitch-to-RxLink
  Become RxLink supervisor, watching for Heartbeat and bit errors.
  Log bit error(s) and disparity error(s) observed on RxLink
  if (˜flag-LaneHealth) OR (flag-ForceRetraining goes 0-to-1) OR (˜flag-Heartbeat)
  Jump to Step-1

2.8.3 Verification Checklist:

1. mis-alignment of rxfclk among group of 8
2. verify framing and loss of framing conditions
3. verify Lane Health Status algorithm
4. verify manual clearing of flag-LaneHealth
5. verify bit errors - invalid characters and disparity errors
6. verify user programmable time delay
7. Set/clear flag-LaneHealthStatus during RxlinkSync from primary input.
8. Asynchronous events flag-ForceRetraining and flag-Heartbeat
9. Set/clear flag-RxLinkSync, flag-MissionMode
10. Enable/disable Rxlc
11. Verify FarEndLoopback mode
12. a. Verify bit-blasting mode
    b. Inject disparity error and invalid character error during bit-blasting mode

2.9 The Fabric Link Transmitter

The Fabric Link Transmitter (TxLink or FLT) design has eight serial transmitter lanes for data packet transfers and one serial receiver lane for control packet transfers. The Txlink is constructed from one LinkEncoder which is comprised of eight LaneEncoders, eight TxLanes, one Rxlane, and one TxLC module, as shown in Figure-2.8.

Data packets originate at the fabric switch and send 64-bit wide FORD to LinkEncoder every sclk. The FORD is segmented into eight lanes, each lane carrying a byte. The lanes are identified from 7 through 0. The LinkEncoder has eight LaneEncoders which are identified as LaneEncoder 7 through 0. The LinkEncoder segments a FORD into eight lanes and transfers a lane to each LaneEncoder. The LaneEncoder performs 8B10B encoding and transfers encoded_data[9:0] to TxLane. There are eight Txlane modules and they are identified from 7 through 0. The TxLane serializes data from LaneEncoder and transmits it on SERDES PHY. Thus the data path originates at fabric switch in byte-x, and then passes through LaneEncoder-x, TxLane-x, and ends at the serial transmitter PHY.

Correspondingly, 8B10B encoded control packets arrive on serial receiver PHY of Rxlane-FC. The Rxlane-FC will de-serializes data and transfers aRxDI[19:0] to Wordsync module which is part of the TxLC module. The

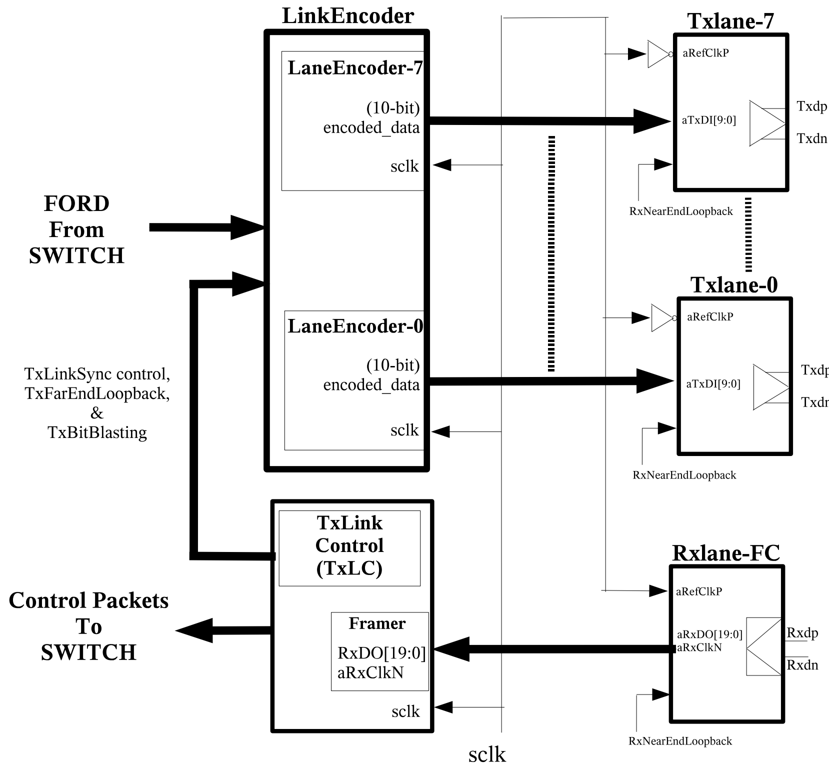


Figure 2.8: Transmitter Link

Wordsync module will decode 8B10B coding and transfer a byte of data every sclk cycle to the fabric switch. TxLC has another output port for supporting loopback operations.

Fabric switch clock, sclk at 200 MHz, is distributed to the TxLC and the LinkEncoder modules as the primary clock. True copy of sclk is distributed to Rxlane-FC on its aRefClkP pin. The inverse copy of sclk is distributed to Txlane modules on their aRefClkP pins as a reference clock.

TxLC has a controller which is responsible for (i) executing hardware routine called TxLinkSync. Upon successful execution of hardware routine TxLinkSync, TxLink has acquired link synchronization. (ii) Upon successful completion of TxLinkSync, TxLC enters the state of MissionMode during which TxLink is functional and switch fabric at both ends of ICE9 can exchange packets. In MissionMode, TxLC will act as a link supervisor and keep checking for link errors including loss of link synchronization. When it detects that link synchronization is lost, then TxLC exits MissionMode and enters hardware routine TxLinkSync for re-synchronization of TxLink. TxLC has another output port for supporting TxLink loopback path.

In hardware execution routine TxLinkSync, controller TxLC communicates with hardware execution routine (called RxLC and described earlier in section-2.8) of corresponding 8-lane receiver using return path through LinkEncoder. In hardware execution routine, TxLC controller acts as a slave and RxLC controller acts as a master. Hardware routine TxLinkSync gets executed once after power is up, and after PLLs are locked, and the reset signal is negated. The TxLinkSync routine is entered from MissionMode if loss of synchronization, i.e. loss of heartbeat, is detected.

Loss of TxLink synchronization will occur when any of the following conditions is detected.
(a) loss of signal on serial receiver or excessive character or disparity errors on fc-lane,
(b) setting of flag-ForceRetraining through SCB (see section-2.9.1),
(c) clearing of flag-Heartbeat from heartbeat timeout on the fc-lane (see section-2.9.1),
(d) disabling TxLink with SCB TxLcControl Ena bit, SoftReset, or hard reset line.

2.9.1 Status Flags required by TxLC

TxLC will have following status flags.

1. Flag-LaneHealthStatus RxLane-FC provides lane status through flag-LaneHealthStatus. The flag-LaneHealthStatus is set if RxLane-FC is receiving valid 8B10B encoded characters from lane, incoming characters are clear of disparity errors, and has acquired frame, otherwise flag is reset. Software may also reset flag-LaneHealthStatus through SCB by setting ClrLaneHealth.

2. Flag-ForceRetraining The TxLinkSync routine may be initiated by Software setting flag-ForceRetraining on SCB. The transition from 0 to 1 of flag-ForceRetraining causes TxLyncSync to be initiated. Software should then clear flag-ForceRetraining so it is available for future use.

3. Flag-Heartbeat During MissionMode the TxLink uses a "heartbeat" method of detecting good communication from the RxLink in the other chip. The following steps describe heartbeat operation:

- When the TxLink achieves MissionMode, flag-Heartbeat is set.
- During MissionMode, the RxLink in the other chip drives continuous control packets which will use SOP (start of packet char, k28.3) character as a marker. The transition from SOP character to non-SOP character detects the heartbeat.
- During MissionMode, if heartbeat is not detected for consecutive 128 clock cycles, then it is assumed that link has lost heartbeat and flag-Heartbeat is cleared, otherwise it remains set.
- A loss of Heartbeat causes a loss of MissionMode, and routine TxLyncSync is re-entered.
- Also, if MissionMode is lost for any other reason, flag-Heartbeat will be cleared.

4. Flag-TxLinkSync The TxLC controller maintains a status flag-TxLinkSync. When flag-TxLinkSync is set, it indicates that TxLC is in hardware execution routine otherwise this flag is reset. This flag will remain set if hardware execution routine TxLinkSync has encountered a failure and/or routine has not completed successfully.

5. Flag-MissionMode The flag-MissionMode is a status flag. It is controlled by TxLC module. This flag is set when hardware execution routine TxLinkSync has been successfully completed. When flag-MissionMode is set, it indicates that TxLink is operational and control of a link has been transferred to the fabric switch. Setting of flag-MissionMode implies that (i) fabric switch at both ends will maintain link heartbeat on data transfer in both direction, (ii) spurious lane errors will be detected by FC lane controller as bit errors, and those errors are logged, (iii) spurious bit errors will not make link unusable, and (iv) persistent bit errors on FC lane will cause loss of link health by resetting of flag-LaneHealth, which in turn, will force re-entry of the hardware execution routine TxLinkSync by resetting flag-MissionMode and setting flag-TRxLinkSync.

2.9.2 TxLinkSync Routine

Jump to Begin:

- Begin:
  If (flag-ForceRetraining) then jump to Step-1
- Step-1:
  Set flag-TxLinkSync, reset flag-MissionMode, reset flag-Heartbeat.
  Force Idle on Control Packet-to-Switch, Disable data path from FORD-to-TxLink, Force k_28.5 on all 8 LaneEncoders (send NULL)
  (sending k_28.5 without Heartbeat will force receiver to jump to RxLinkSync routine)
  Wait till flag-LaneHealthStatus is set, then jump to Step-2
  Note: If flag-LinkHealth is asserted for less than 3-ticks, then controller will not jump to Step-2 and will re-enter or remain in Step-1. For each occurance of such case, or each jump to Step-2, R_FltxTxLcCount will be incremented.
- Step-2:
  (when flag-LaneHealth is set, then control lane is receiving k_28.5)
  If (˜flag-LaneHealth) then jump to Step-1
  else

Force Idle on Control Packet-to-Switch, Disable data path from FORD-to-TxLink, Force k_28.5 on all 8 LaneEncoders (send NULL)
(sending k_28.5 without Heartbeat will force receiver to jump to RxLinkSync routine)
Wait for time T1 = (R_FltxTxLcControl.Step2WaitTime number of sclks), where
T1(min) = (Rate of Heartbeat + 4 times maximum link delays) = (100 * sclk period) + 4 * 10nS = 500 + 40 = 540nsec
Jump to Step-3

- Step-3:
  If (˜flag-LaneHealth) then jump to Step-1
  else
  Force Idle on Control Packet-to-Switch, Disable data path from FORD-to-TxLink, Enable FarEndloopback path
  Jump to Step-4
- Step-4:
  If (˜flag-LaneHealth) then jump to Step-1
  else
  If EOLS (k28.2) then Disable FarEndloopback path, set flag-Heartbeat, and jump to END
  else
  Remain in Step-4, continue FarEndloopback, keep watching for EOLS or ˜flag-LaneHealth (no time limit).
- END: (enter MissionMode operation)
  Set flag-MissionMode, Reset flag-TxLinkSync
  Enable data path from FabricSwitch-to-TxLink.
  Enable control packet path from TxLink-to-FabricSwitch
  Become TxLink supervisor, watching for Heartbeat and bit errors.
  Log bit error(s) and disparity error(s) observed on TxLink
  if (˜flag-LaneHealth) OR (flag-ForceRetraining goes 0-to-1) OR (˜flag-Heartbeat)
  Jump to Step-1

2.9.3 Verification Checklist:

1. Set/clear flag-LaneHealthStatus during TxLinkSync
2. asynchronous events flag-ForceRetraining, flag-Heartbeat
3. set/clear flag-TxLinkSync, flag-MissionMode
4. Enable/disable TxLC
5. Verify FarEndLoopback mode
6. a. Verify bit-blasting mode
   b. Inject disparity error and invalid character error during bit-blasting mode

2.10 Reset bring-up sequence

Following steps are required to bring-up link after reset:
(later on we say what to do to cause each of these steps)

1. Wait for refclk stabilization time = TBD
2. Wait for QPMA Tx PLL(s) lock (aTxClkP stabilization time) = 15 uS
3. a. Wait for calibration time = TBD
   b. Wait for QPMA Rx PLL(s) unlock = 10 uS
   c. Wait for QPMA Rx PLL(s) lock (aRxClkN stabilization time) = 15 uS
   (AnalogBits “ABIPCCE2 Custom PLL DATASHEET”
   says 10 uS is enough, but we’ve seen it take slightly longer to lock)

4. Wait for Skip-beat operation. Max Time it takes =
   a. max. skipbeat operation before step_step = 5 x (31 x 5) = 775ns
   b. skipbeat operations before second_step = 5 x (31 x 5) = 775ns
   c. skipbeat operations before final_step = 5 x (31 x 5) = 775ns
   d. skipbeat operation at adjustment = 5 x (31 x 5) = 775ns
   e. Total time = 775 x 4 = 3100ns

5. Wait for framing. Max time = 3200ns as described in section on framing.

6. Now PMA is ready and LinkSync can begin

7. When link enters MissionMode, invalid character error and disparity error counters may contain non-zero values. Software must initialize these registers before enabling interrupt from these registers.

2.10.1 When do Link Registers Get Reset

2.10.1.1 AnalogBits QPMA Registers

AnalogBits Internal QPMA Registers only go to their "reset values" or more-accurately, their "power-on values" when the power gets turned on. They are unaffected by either SoftReset registers or the "hard reset" reset signal coming into FL. This refers to the registers that are within the AnalogBits QPMA's, not the QSC registers. These registers are not directly accessible from the SCB bus.

2.10.1.2 QSC Registers

QSC Registers are reset by the "hard-reset" reset signal, but are not affected by any SoftReset registers. Most of the QSC Registers (with the exception of R_QscInterrupt) are used to allow indirect access from the SCB bus to the AnalogBits Internal QPMA Registers.

2.10.1.3 FLT and FLR link Registers

Flt0, Flt1, Flt2, Flr0, Flr1, Flr2 Registers get reset by the "hard-reset" reset signal, but are not affected by any SoftReset registers. The SoftReset bit of a particular link affects operation of that particular link only.

For example, writing a 1 and then writing a 0 to R_Flr2SoftReset will cause all the control circuitry of Flr2 to go to their reset values including resetting all the internal state machines of link FLR-2. This will not cause any of the R_Flr2* registers to go to reset values.

2.10.2 Enabling Links

When power first comes on the Links are disabled and non-operational in several ways: QPMA units do not have valid Impedance Settings, QPMA units do not have valid Calibration Settings, and Link units have their LinkSync Routines disabled.

After a hard-reset, or SoftReset, if configuration had previously been done during this period of power being ON, the QPMA units retain their prior Impedance and Calibration Settings, but the Link units have their LinkSync Routines disabled.

The recommended steps to being up links are :

1. Determine QPMA Impedance Settings, to "factory values", or discover them.

2. Configure QPMA Calibration Settings, to saved values, or discover them.

3. Initialize SkipBeat Functions.

4. Enable Links.

2.10.2.1 Determine QPMA Impedance Settings

Since these settings only depend on the silicon manufacturing process of that particular ICE-9's individual QPMA cores, not which slot the board it's on is plugged in to, the best settings can be determined at the factory, saved somewhere, and loaded at this time.

If we choose to discover them each time we power-on, or when they're determined "at the factory", they can be determined by a process of SCB bus writes and reads, without the link being enabled. The status of ICE-9's at the other end of Links doesn't matter.

Ques: what about that link going to a slot with missing board?

QPMA Impedance values are set using SCB register R_QscQpmaImpCalibration.The process of discovering correct values also uses SCB register R_QscQpmaStatus. Refer to Analogbit's PRM manual for further detatils.

2.10.2.2 Configure QPMA Calibration Settings

QPMA Calibration Settings for a particular Link must be determined while that Link is enabled by the procee-dure above, and correct Impedance Settings must already be loaded.

QPMA Calibration Settings for a particular Link must be determined by trial and error during the same time period that the ICE-9 at the other end of that Link's fabric connection is also trying to determine it's own QPMA Calibration Settings for Link on that fabric connection.

QPMA Calibration values are set using R_QscGo, R_QscStatus, R_QscCA, R_QscSerDatAR, R_QscSerDatT, R_QscSerDatP.

Note that the details of "good working algorithm for trial and error", and what values to try, are not listed here yet.

When both ends of a Fabric Connection have configured good QPMA Calibration values, each end can see that because the LinkSync routine will make progress to later steps. This can be seen in R_FlrxRxLcStatus and R_FltxTxLcStatus registers by looking at fields Steps and MissionMode.

The ICE-9's on the two ends of a particular Fabric Connection may be beginning this step at significantly-different times, differing by thousands of clocks. The early steps of the LinkSync Routines don't mind this, don't time-out, and will have no problem waiting for the other end to start trying Calibration Values. Similarly, the algorithm for trying values and checking LcStatus will be a repeating loop, continuing long enough for the other ICE-9 to start trying values.

2.10.2.3 Initialize SkipBeat Functions

For the 3 FLT's, write 1, and then write 0 to bit SkipBeatEnable in R_FltxFcLaneControl, leaving field Skip-BeatOffset at it's reset value (unless it has been determined that another value should be used). If QPMA PLLs are locked on stable clocks, the SkipBeat function is fast enough to complete before you can get the 0 written.

For each lane in each of the 3 FLR's write 1, and then write 0 to bit SkipBeatEnable in the R_FlrxLaneControl register for that lane, leaving field SkipBeatOffset at it's reset value (unless it has been determined that another value should be used).

2.10.2.4 Enable the Links

A Link is enabled by writing 3 times to it's "LcControl" register, first to enable it, then to set the ForceRT bit, then to clear the ForceRT bit. Write 0x2, then write 0x3, then write 0x2 to each of R_Flt0TxLcControl, R_Flt1TxLcControl, R_Flt2TxLcControl, R_Flr0RxLcControl, R_Flr1RxLcControl, R_Flr2RxLcControl.

Note: All interrupt enables reset to a not-enabled state. If since the last reset interrupts have been enabled, it is desirable to disable interrupts from links which are about to enter in ForceRT. Also, before enabling link, it is desirable to clear all interrupts from that link, verify that all interrupt generating conditions are not present.

2.11 Diagnostic Modes

The diagnostic modes are supported to aid in lab debug of links. It is a requirement that for correct operation in diagnostic mode, the receiver link and the transmitter link at both ends of a link have successfully configured their respective QPMA calibration settings. Also, at most only ONE of the 3 diagnostic modes described below

(Near End Loopback, Far End Loopback, or Bit-Blasting Mode) should be enabled at any one time for a particular link or pair of links.

2.11.1 NearEndLoopback Mode

The NearEndLoopback mode of opeartion is supported to verify that receiver path is connected to transmitter path and thus verify data path from FSW to transmitter link to receiver link to FSW.

In NearEndLoopback mode, a receive data link is connected to a transmit data link and thus receiver lanes are disconnected from off-chip path from PHY. It is important to note that the transmitter lanes will still drive transmitter PHY.

All 3 links can be simultaneously configured in NearEndLoopback Mode.

2.11.1.1 Link-0

For connecting FLR0 to FLT0, set LpBkNearEnd[3:0] field of R_QscQpmaControl0 and R_QscQpmaControl1 register. Also set LpBkNearEnd[0] field of R_QscQpmaControl6.

2.11.1.2 Link-1

For connecting FLR1 to FLT1, set LpBkNearEnd[3:0] field of R_QscQpmaControl2 and R_QscQpmaControl3 register. Also set LpBkNearEnd[1] field of R_QscQpmaControl6.

2.11.1.3 Link-2

For connecting FLR2 to FLT2, set LpBkNearEnd[3:0] field of R_QscQpmaControl4 and R_QscQpmaControl5 register. Also set LpBkNearEnd[3] field of R_QscQpmaControl6.

2.11.2 FarEndLoopback Mode

The FarEndLoopback mode of operation is supported to verify that receiver link is loopbacked to transmitter link in SCLK domain, i.e. Flt0 and Flr0 can be connected, and/or Flt1 and Flr1 can be connected, and/or Flt2 and Flr2 can be connected.

Do not confuse this with the so called "FarEndLoopback" used in the LinkSync Routine, which is within an individual FLT, FC lane in to the 8 data lanes out.

This FarEndLoopback mode is supported to verify 8 data lanes and 1 flow control lane connectivity from receiver PHY to transmitter PHY in SCLK domain. The far end loopback path will bypass 10B8B decoding at the receiver end and 8B10B encoding at the transmitter end.

The far end loopback mode will not invoke skipbeat function, or acquire lane health, or word synchronization.

The far end loopback mode assumes that impedance and calibration circuit for a given link is initialized to correct settings and skipbeat function for that link is completed successfully.

In FarEndLoopback mode, the mission mode signal going to FSW is de-asserted and thus FSW is disconnected to/from PHY or fabric switch is bypassed. This would allow the 2 remote Ice9's that have been connected to each other through this local Ice9 to bring up MissionMode with each other through this 2-hop link.

All 3 links can be simultaneously configured in FarEndLoopback Mode.

2.11.3 Bit-Blasting Mode

The bit-blasting mode is supported to verify link integrity from a transmitter to a receiver. For a given link, it is suggessted that bit-blasting mode may be invoked only after both ends of a link have entered in Mission Mode. The bit-blasting mode does not attempt to invoke skipbeat function, or does not attempt to acquire lane health, or word synchronization.

Each link may be configured in Bit-Blasting Mode as follows:

1. Verify that Link under test is in mission mode.
   This step is not mandatory step but it is strongly suggested because for bit-blasting function to operate correctly, lane must have successfully completed skipbeat function and acquired lane health. Note that bit-blasting mode does not attempt to invoke skipbeat function, nor does it attempt to acquire lane health, nor does it acquire word synchronization.

2. Write 0x8800 in R_FlrxBBDiag register when FLR is to enter in bit-blasting mode (Refer to section-2.17.18). Write 0x80FF in R_FLtxBBDiag register when FLT is to enter in bi-blasting mode (Refer to section-2.16.16). This step serves 3 purposes:
(a) It disables heartbeat counter from expiring which disables invoking hardware LinkSync routine.
(b) It disconnects FSW from link by deasserting MissionMode and DataValid signals going to FSW.
(c) It sends NULL and ANULL characters on all driver lanes which keeps other end of link in MissionMode.
This will allow software to manage bit-blasting mode at both ends of link with ease.

3. Write R_FlxxDiag register keeping BBMode set and also selecting other fields of this register.
Note that driver lanes which are not selected to drive bit-blasting pattern will drive PNULL (k28.5) patterns.

4. Monitor R_FlxxDiagStatus (section-2.17.19 and section-2.16.16) register for results of bit-blasting mode.
There are 2 bits assigned per receiver lane. One bit is Sync-bit and it indicates if receiver lane has acquired synchronization for configured bit-blasting pattern, and the other bit is Error-bit which indicates if any error is observed after synchronization is acquired.
By de-selecting lane, both status bits associated with this lane are cleared. By selecting lane again will make both status bits assocaited with this lane valid. While in BBMode, toggling of lane select field is permitted.

5. Before exiting bit-blasting mode, execute above step-2.

6. Clear R_FlxxDiag register to enter in MissionMode again.

2.11.4 ATE Testing of Analogbits ABICDR43

ATE testing of Analogbits ABICDR43 macro can be carried out at speed and in Near-End-Loopback mode. Instructions for Near-End-Loopback test follows.

1. Execute reset power on sequence in ICE9.

2. Put all seven QPMA in NearEndLoopback mode by writing to R_QscQpmaControl registers.
(LpBkNearEnd=1, ForceTxHiZ=1, ForceRxHiZ=1, and clearing rest of the bits)

3. Initiate SkipBeat function in FLR0, FLR1, and FLR2 receiver links. Also initiate Skipbeat function in FLT0, FLT1, adn FLT2 links.
(toggle SkipBeatEnable bit in all LaneControl registers)

4. Initiate LinkSync routine in FLR0, FLR1, and FLR2 receiver links. Also initiate Skipbeat function in FLT0, FLT1, adn FLT2 links.
(set Ena bit and then toggle ForceRT bit in all LcControl registers)

5. Wait for 10 microsec (enough time for links to reach MissionMode).

6. Read link status register of FLR0, FLR1, FLR2, FLT0, FLT1, and FLT2 and verify that each link (a) is not in reset, (b) is in MissionMode, (c) has heartbeat, and (d) has its Step[3:0] field clear.

2.11.5 PLL Bypass Mode Testing of Analogbits ABICDR43

Analogbits ABICDR43 serdes macro (QPMA) has 5 seperate PLL. One is TXPLL and used by four transmitter lanes. The other four copies are CDRPLL and each receiver lane uses one copy. The PLL Bypass Mode test should configure all five PLLs of QPMA in bypass mode and then validate data path connectivity from transmitter lane to corresponding receiver lane. When PLL are in bypass mode, it generates internal high speed clock same as that of reference clock. Also, this test is intended to be used for structural testing of serializer and deserializer of QPMA.

In PLL Bypass test, once QPMA is configured in PLL bypass mode, the data pattern of all 1's is driven on its parallel port TxDI[9:0] and kept unchanged for 100 sclk cycles. Data from parallel port go through serializer of transmit path, then loops back because of near end loopback, and then gets deserialized in receiver path on subsequent clock cycles. After "TBD" sclk cycles (but less than 100) cycles later it settles down on receiver parallel port RxDO[19:0]. Test will check if receiver port has observed all 1's on all outputs. Test is decalred partially successful if all 1's are observed on RxDO[19:0].

Two more test loops as described above are carried out, first for data pattern of 0's and next one for 1's. Test is declared successful only if all 3 data patterns are successfully observed on output port.

There are 7 instances of QPMA in ICE9. Following steps are recommended for testing PLL in bypass mode of each QPMA.

1. Configure TXPLL in reset by setting bits TxPllRst of R_QscQpmaImpCalibratio.
2. Configure CDRPLL in reset by setting bits of CDRPLLRst of R_QscQpmaControl.
3. Configure QPMA in power-up mode by clearing bit so f RxPwrDown of R_QscQpmaControl.
4. Disable IDDQ mode of QPMA by clearing IDDQ bit of R_QscQpmaControl.
5. Force transmit and receive macro in HiZ by setting ForceTxHiZ and ForceRxHiZ bits of R_QscQpmaControl.
6. Enable near end loopback by setting LpBkNearEnd bits of R_QscQpmaControl.
7. Enable high frequency transmit and receive clock by asserting TxHFClkDnB and RxHFClkDnB of R_QscQpmaTestControl.
8. Wait for 400 sclk cycles (enough time for data input patterns to propagate from TxDI to RxDO register) and then check if PllBpStatus bits R_QscQpmaStatus to verify test result.

2.12 Error recovery procedure

Fabric link CSRs are designed to capture and hold cause and state of the error. These status registers are cleared by SCB master. The SCB master should clear error state and error status register(s) before reverting to normal mode of operation.

2.12.1 Force Retraining

The ForceRT bits of csr-2.16.10 and csr-2.17.10 allow forcing retraining sequence on respective controller. The retraining routine should be forced by SCB master only after clearing of all error states in respective controller. The retraining routine can also be forced while respective controller is in Mission Mode.

2.13 Bring-Up Failure Points

The link bring-up process can fail at a variety of detectable points. Here is a list of them, and what it may mean if you fail at each point. Possible example define-names are given in all-capitals for each failure.

The first few are "whole-node", and later ones are "for a link".

For a given FLR or FTL, these are listed in the same order as the actions (and checks) are done, doing first the whole-node actions, then the actions for the given FLR or FLT. So, if you fail at a particular point in this list, that means all previous actions for that link were successful. (exception: ERR_FL_INIT_CODE_<n>)

In a failure, it would be nice also say what the other end is, which board/node/link, or have system-sensitive diags function like "print_link_other_end(this_board, this_node, this_link)".

==

These first 4 failure points have to do with calibrating the 7 qpmas:

ERR_FL_TXPLL_NO_LOCK = Not all of the Tx PLLs locked. Specifically, failed to get R_QscQpmaStatus.TxClkStable in at least 1 of the 7 qpma's, after waiting long enough after setting and then clearing the 7 R_QscQpmaImpCalibration.RxPllRst bits.

Look at whether you've started chip clocks and voltages correctly. If you still get this, you probably have a bad Ice9 chip (bad Tx PLL).

Note that if you are using chips that had the normal testing at the chip vendor, the packaged chips have been tested for this being good. The same is true for failures below where "bad chip" is likely cause.

ERR_FL_ZCALIB_TOO_HI = The determined ZCalib transition point was above the legal range, or ZCompOp was 1 no matter how high a ZCalib was tried, on at least one qpma.

ERR_FL_ZCALIB_TOO_LO = The determined ZCalib transition point was below the legal range, or ZCompOp was 0 no matter how low a ZCalib was tried, on at least one qpma.

Out-of-legal-range ZCalib transition point in one qpma in an Ice9 supplied proper voltages and clocks, indicates a bad Ice9 chip.

It's nice if values, good or bad, go into a log file somewhere, in case in a later step we have excessive bit errors we're trying to diagnose.

ERR_FL_QSC_WR_FAIL = When setting qpma lanes with A T P R values, at least one lane didn't get QscSuccess within a reasonable time.

Since these are done individually, the software knows which ones failed. Repeated failure is due to bad chip, incorrect software sequence or Qsc addresses, or inadequate wait time.

Once you've calibrated the 7 qpmas, you can bring-up (or not bring up) each of the 3 FLTs and each of the 3 FLRs separately. If you don't have working Ice9s at the other end of some of these 6 links, the others can still be brought up to MissionMode, and transfer packets.

The following errors must be clear whether it's FLR or FLT, and which link. This information could be made part of the error-define-name, or be provided as extra information.

ERR_FL_RXPLL_NO_UNLOCK = After resetting Rx PLLs for this link's lanes, one or more failed to unlock in a reasonable time. (FLT has only 1 Rx Lane)

You should be able to unlock PLLs no matter what the Ice9 at the other end is doing. Failure here suggests this Ice9 is bad, or it has bad configuration/clocks/voltages.

ERR_FL_RXPLL_NO_LOCK_SOME = In this FLR, after unresetting the 8 data lane Rx PLLs, some failed to lock onto incoming signals in a reasonable time.

ERR_FL_RXPLL_NO_LOCK_ALL = In this FLR, after unresetting the 8 data lane Rx PLLs, all 8 failed to lock onto incoming signals in a reasonable time.

ERR_FL_RXPLL_NO_LOCK = In this FLT, after unresetting the control lane Rx PLL, it failed to lock onto incoming signal in a reasonable time.

As shown above, for FLR I suggest writing the small extra code to differentiate between "all failed to lock" and "some failed to lock" because "all failed to lock" strongly suggests that the Ice9 at the other end has not completed initial calibration, is in reset, or there's actaully NO Ice9 at the other end.

Rx PLL locking is the first point in the process where we are affected by the Ice9 at the other end. Rx PLL locking is also the first point in the process where we are affected by bad connections, serious noise on the fabric between chips, or improper calibration on either end.

Failure to get Rx PLL lock is a condition worse than the "bit errors" which can cause problems in later steps. The following can interfere with Rx PLL lock, or cause bit errors and prevent LaneHealth:

- Other end is not yet transmitting.
- Other end has not finished calibration.
- Other end has wrong Tx calibration.
- This end has wrong Rx calibration.
- This Ice9 or other-end Ice9 is in some diagnostic mode (see next section "Registers that can Prevent Link Coming Up").
- Signal not strong enough, serious reflections, or noise from outside of the differential pair (wrong Tx or Rx calibration).
- Unstable Tx clock, other-end Tx PLL has not locked, or other-end sclk not stable.
- One or both signals of differential pair have a bad connection.
- Bad capacitor on differential pair
- Other end is in reset.
- Other end has power problems.
- Other end is on a board that's not plugged-in.
- Bad Ice9 on either end.

Note that you can get "false Rx PLL lock". Reset then unreset of PLL is done to clear old false locks. If the signal on the differential pair is very bad, has data plus lots of noise, or not being driven but wires are picking-up noise, the Rx PLL might still lock onto what it sees.

You might check whether PLL locking is coming and going.

If it's an FLR, Diagnostics should say WHICH rx lanes failed to lock.

ERR_FL_SKIPBEAT_FAIL = SkipBeat failed for at least one Rx data lane, when checked more than long enough after SkipBeat init.

SkipBeat is aligning a divided-by-10 version of the clock formed by Rx PLL locking, with Ice9's sclk. The most likely reason for SkipBeat failure is that you no-longer have Rx PLL lock, or it comes and goes.

A valid SkipBeat is one that completes near the specified time period. Waiting far longer and eventually seeing SbSuccess is not valid. You should restart SkipBeat AFTER aquiring a consistent Rx PLL lock, then look for success.

ERR_FL_NO_HEALTH = No LaneHealth on at least one Rx lane, when checked a sufficient time after getting SkipBeatSuccess on all Rx lanes. (FLT has only 1 Rx Lane)

The hardware will continuously try to get LaneHealth, with no software-starting of Rotator needed.

Do you still have Rx PLL lock? Even if you do, try re-doing Rx PLL lock and SkipBeat.

Some reasons listed under ERR_FL_RXPLL_NO_LOCK can prevent LaneHealth, even if we have Rx PLL lock.

With a weak signal or noise causing bit errors in an ongoing manner, there may be enough edges with equal spacing to sustain Rx PLL lock, while causing enough character errors to prevent LaneHealth, or cause LaneHealth to come and go.

With LaneHealth==0 you should see Rotator trying different values, and character error counts increasing.

A miss-match of configured sclk frequencys between 2 Ice9's may go unnoticed up to this point, where you are unable to get LaneHealth.

Do we have R_QscQpmaStatus.RefClkStable at both ends?

Is this Ice9 or other-end Ice9 in some diagnostic mode? See next section "Registers that can Prevent Link Coming Up".

If it's an FLR, Diagnostics should say WHICH rx lanes can't get LaneHealth.

At this point the FLT or FLR is started into the LinkSync routine, which will try repeatedly to go through Step1, Step2, Step3, Step4, to MissionMode, unless, in uncommon cases, it gets stuck at some step. See ERR_FL_MISL_GONE below for more details on getting stuck in Steps.

ERR_FL_SYNC_BIT_ERRS = Bit errors on at least one lane in this link, even though LinkHealth is good, while waiting to get to MissionMode.

After getting LinkHealth (same as LaneHealth on all lanes), all bit error counters should be cleared. This condition is that you got new bit errors, after clearing the counters.

If the number of bit errors is unchanging, wait awhile and the link may still achieve MissionMode.

ERR_FL_SYNC_LOST_HEALTH = LaneHealth is false on at least one lane in this link, while waiting to get to MissionMode.

The hardware will try to recover LaneHealth, and if it can, the link may still make it to MissionMode.

ERR_FL_SYNC_TIMEOUT = MissionMode not achieved on this link after too long a time in state sc-fab_link_state_syncing.

See ERR_FL_MISL_GONE below for more details on different cases.

If bit errors are not changing, LaneHealth is good, something's wrong, Link may be stuck, and may need to be restarted by Software.

The error codes below are for after MissionMode has been achieved. Using different error-defines after Mission-Mode gives a little more information,

If MissionMode is lost, the hardware will repeatedly try to recover through the LinkSync routine to MissionMode, unless it gets stuck.

You can read FlrxRxLcStatus or FltxTxLcStatus to see many aspects of link state in one register-read: Mission-Mode, LinkSync-active, LinkSync Step, whether all Rx PLLs are locked, whether all lanes have LaneHealth.

The following error cases are listed from lightest-to-heaviest badness:

ERR_FL_MISL_BIT_ERRS = After MissionMode achieved, MissionMode stays up adequately, but bit errors keep happening on this link.

ERR_FL_MISL_LCCOUNT_HI = MissionMode is up now, but it comes and goes too often.

This error is being reported because ”too many” past reads of LcStatus.MissionMode gave 0, or because FlrxRxLcCount or FltxTxLcCount has become ”too high”, or we see that count continue to increment.

——————————————————————————-

ERR_FL_MISL_GONE = After MissionMode achieved, MissionMode is now false in this link. Rx PLLs are still locked. Rx Lanes have LaneHealth now.

Poll for MissionMode for enough time for Syncing to happen. If still no MissionMode after reasonable time, Software should re-start the LinkSync routine.

If Steps==1 and is unchanging, or repeately going back to Steps==1, it may have excessive bit errors. Check all lanes for LaneHealth, and see if BitErrors are changing.

If Steps==2, or a mix of Steps==1 and Steps==2 for a long time, we have a low but consistent rate of bit errors, preventing Syncing.

In an FLR, if Steps==4 (Step3), with FlrxRxLcStatus.RxLinkSync==1 for a long time, the FLT at the other end should be looked at. It's either getting lots of bit errors over the control lane from this FLR, or the control lane seems dead, or FLT hasn't been started to do Syncing.

In an FLR, if Steps==8 (Step4) for too long, we have ”the Step4 hang” due to infrequent bit errors, and Software must restart.

It's ok for an FLT to have Steps==8 (Step4) for a fairly long time, while the FLR at the other end tries repeatedly to get MissionMode. But it also might be that the FLR is stuck or has a false-MissionMode. If excessive time passes, the FLT can try a restart of LinkSync, which can clear some conditions. If it doesn't, the problem must be dealt with at the other end, in the Ice9 containing the FLR. At the other end you can see if FLR is stuck (requiring a restart), or one or more of the data lanes from FLT is having excessive bit errors, or seems dead.

After MissionMode, in FLT and FLR, if no SoftReset has been done, either

(a) MissionMode==1, LinkSync==0, Steps==0, or

(b) MissionMode==0, LinkSync==1, Steps== one of 1, 2, 4, 8.

Any other combination is a (rare) corrupt state, and you should restart the link, doing SoftReset first. After SoftReset expect MissionMode==0, LinkSync==0, Steps==0.

——————————————————————————-

ERR_FL_MISL_NO_HEALTH = After MissionMode achieved, MissionMode is now false in this link. Rx PLLs are still locked, but LcStatus.LinkHealth==0 consistently with repeated reading, which means at least one Rx Lane has LaneHealth==0.

——————————————————————————-

ERR_FL_MISL_RXPLL_NO_LOCK = After MissionMode achieved, MissionMode is now false in this link. AllReset (or AllRxLanesReset) is 0 which means at least one Rx Lane has lost PLL lock.

ERR_FL_MISL_RXPLL_NO_LOCK_ALL = In this FLR, after MissionMode achieved, MissionMode is now false in this link. AllRxLanesReset is 0, but furthermore, all 8 bits of FlrxLinkStatus.CdrPllLock are 0, consistently. This suggests a shut-down or removal of the Ice9 at the other end.

==

ERR_FL_INIT_CODE_<n> = Link bring-up failed in one of the places where a software sanity check is done. This failure had nothing to do with hardware behavior. <n> is a unique number for each such place in the link bring-up code.

2.14 Registers That Can Prevent Link Coming Up

After any diagnostic or manual mode has been used, like bit-blasting or loopback, you need to either restore all registers to their normal values or do a hard-reset before attempting normal bring-up. Bring-up software typically doesn't write reset values to registers it would not otherwise be writing. Even when bring-up software writes a register, that software may be carefully leaving-unchanged fields within a register it's not actively using.

Abnormal configuration in an Ice9's registers (or the Ice9 at the other end of the Link) can prevent a Link from coming up to MissionMode.

These registers (or specific fields) could prevent bring-up if badly configured:

——————————

R_FltxSoftReset
R_FltxFcLaneControl (fields: ForceSkipBeat, SkipBeatOffset)
R_FltxAltNull
R_FltxHeartbeat (fields: Dis, Threshold)
R_FltxS2WaitTime

R_FltxMOR
R_FltxFarEndLoopback
R_FltxBBDiag

R_FlrxSoftReset
R_FlrxWSyncMode
R_FlrxHeartbeat (fields: Dis, Threshold)
R_FlrxS2WaitTime
R_FlrxLaneControl[7:0] (fields: ForceSkipBeat, SkipBeatOffset)
R_FlrxMOR[7:0]
R_FlrxBBDiag

R_QscQpmaControl[6:0] (all fields other than CDRPLLRst)
R_QscQpmaTestControl[6:0]

Also, if you can't bring up a link because of excessive interrupts, maybe a link interrupt is inappropriately enabled, or wasn't cleared.

2.15 Common Registers and Definitions

2.15.1 Package Attributes

Package

chip_fl_spec

Attributes

–public_rdwr_accessors

2.15.2 Definitions

Defines

FL

Constant	Mnemonic	Definition
10'h7f	STEP2_WAIT_TIME	Sleep timer value. Cycles to wait in step2.

2.15.3 Link Symbols

Enum

FlSymbols

Constant	Mnemonic	(Code Name)	Definition
8'h1c	ANULL	k28.0	Alternate Null.
8'h3c	SOLS	k28.1	Start of Link Sync.
8'h5c	EOLS	k28.2	End of Link Sync.
8'h7c	SOP	k28.3	Start of Packet.
8'h9c	EOP	k28.4	End of Packet.
8'hbc	PNULL	k28.5	Primary Null.
8'hdc		k28.6	Reserved.
8'hfc		k28.7	Reserved.
8'hf7		k23.7	Reserved.
8'hfb		k27.7	Reserved.
8'hfd		k29.7	Reserved.
8'hfe		k30.7	Reserved.

2.15.4 Flr Events

The following events are trackable by SCB statistical event counting.

Enum

FlrScbEvent

Attributes

-descfunc

Constant	Mnemonic	Definition
8'h00	CYCLES	Sclk cycles. Always counts.
8'h01-8'hff		Reserved.

2.15.5 Flt Events

The following events are trackable by SCB statistical event counting.

Enum

FltScbEvent

Attributes

-descfunc

Constant	Mnemonic	Definition
8'h00	CYCLES	Sclk cycles. Always counts.
8'h01-8'hff		Reserved.

2.16 FLT Registers

2.16.1 R_FltxSoftReset

Register

R_FltxSoftReset

Attributes

-kernel

Address

0x0_0000_0000 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
0	SoftReset	RW	0		Reset Link when set. When written 1, transmitter link remains in reset state. When written 0, the transmitter link logic come out of the reset state.

Operation of SoftReset

When SoftReset is asserted, all CSRs of FLTx remain unaffected by SoftReset. However, control flops within FLTx module are intialized to power-on reset value. After de-assertion of SoftReset, software will have to initiate skipbeat function on its flow control lane and then enable transmit link.

2.16.2 R_Fltx FC Lane Control Register

Register

R_FltxFcLaneControl

Attributes

-kernel

Address

0x0_0000_0004 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
7	ClrLaneHealth	RW	0		Clear lane health. For every transition of 0-to-1 of this bit, lane health bit of FC lane is cleared.
6					Reserved.
5	ForceSkipBeat	RW	0		Force Skipbeat. This bit must remain clear when SkipBeatEnable is clear. When SkipBeatEnable is set : For every transition of 0-to-1 of this bit, RxClk offset is skipped 1-bit time. This field is intended to be used in manual setting of RxClk. This bit should be clear after manual setting of RxClk is completed.
4	SkipBeatEnable	RW	0		Skip Beat Enable. At the transition from 0-to-1, SkipBeat function is executed once using value selected in "SkipBeatOffset". To initialize Skip Beat function, write 1 followed by write 0. For manual setting of skipbeat, write 1, then use ForceSkipBeat (above), then write this bit 0.
3:0	SkipBeatOffset	RW	0x5		SkipBeat Offset. The receiver RxClk offset is equal to "SkipBeatOffset" bit-time wrt sclk. The power-on default value is 5(hex). This field is 4-bit wide and SkipBeatOffset can be selected from 0(hex) to 9(hex). The values in this field are modulo-10. For applying newer value of SkipBeatOffset, SkipBeatEnable should be toggled.

Operating modes of Skipbeat function

At the end of reset sequence, SkipBeatOffset field value defaults to 0x5. It holds offset value in bit-time. At 200Mhz of sclk, bit time is 0.5nsec.

SCB master can modify SkipBeatOffset value and invoke skipbeat function by toggling SkipBeatEnable bit once. This method triggers skipbeat function with selected SkipBeatOffset value. Please note that during this process, if any time reset sequence is invoked then SkipBeatOffset will be defaulted to 0x5.

For manual SkipBeat setting, set SkipBeatEnable=1, (the SkipBeat function will run, completing faster than you can do your next register access), then use single step (sample and move) manual skipbeat algorithm. To do this, repeatedly (a) sample state of "SbTestaRxClkN" of R_FltxLaneStatus register, and (2) move phase of receiver clock 1-bit time by toggling "ForceSkipBeat". Note where SbTestaRxClkN transitions 0-to-1 and 1-to-0, then do additional skips to position the offset correctly relative to those transitions. After desired phase alignment of receiver clock is achived, SkipBeatEnable bit should be cleared.

2.16.3 R_Fltx Lane Status

Register

R_FltxLaneStatus

Attributes

-noregtestcpu -kernel

Address

0x0_0000_0008 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
23:16	PllLock	R	x		Pll Lock status. Holds lock status of 8 Tx PLLs of QPMA module.
15:12	Rotator	R	x		Fc lane rotator value.
11:10					Reserved.
9	LaneHealth	R	x		FC lane health status.
8	FcPllLock	R	x		FcPLL lock status. Holds lock status of fc PLL of QPMA module.
7					Reserved.
6	SbTestaRxClkN	R	x		Test aRxClkN signal. It holds sampled value of aRxClkN signal from Qpma.
5	SbSuccess	R	x		SkipBeat Success. It indicates status of last skipbeat operation. When set, indicates that SkipBeat function has been successful.
4	SbActive	R	x		SkipBeat Active. When set, indicates that SkipBeat operation is active.
3	SbFirstSearch	R	x		State of SkipBeat First search function. When set, indicates that the First Search is completed.
2	SbSecondSearch	R	x		State of SkipBeat Second search function. When set, indicates that Second search is completed.
1	SbFinalSearch	R	x		State of SkipBeat Final search function. When set, indicates that Final search is completed.
0	SbAdjust	R	x		State of SkipBeat Adjust function. When set, indicates that Adjustment is completed.

2.16.4 R_FltxInvCFc

Register

R_FltxInvCFc

Attributes

-kernel -writeonemixed

Address

0x0_0000_000c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
18	Intr	RW1C	0		Invalid Character error interrupt from FltxInvCFc. This bit is set if IntEna is set AND (Compare == Counter).
17	IntEna	RW	0		Invalid Character error interrupt enable for FltxInvCFc.
16	Wrap	RW	0		Enable wrap mode for FltxInvCFc. When set, Counter wraps on maximum count.
15:8	Compare	RW	0		Invalid character error counter comparator for FltxInvCFc.
7:0	Counter	RW	x		Invalid Character error counter for FltxInvCFc. Counts up when invalid charater error is detected on lane. Wraps on maximum count of 8'hFF if Wrap is set. Note: Counter does not count up in clock cycle in which FltxInvCFc is being read or written to by SCB.

2.16.5 R_FltxDispFc

Register

R_FltxDispFc

Attributes

-kernel -writeonemixed

Address

0x0_0000_0010 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
18	Intr	RW1C	0		Disparity error interrupt from FltxDispFc. This bit is set if IntEna is set AND (Compare == Counter).
17	IntEna	RW	0		Disparity error interrupt enable for FltxDispFc.
16	Wrap	RW	0		Enable wrap mode for FltxDispFc. When set, Counter wraps on maximum count.
15:8	Compare	RW	0		Disparity error counter comparator for FltxDispFc.
7:0	Counter	RW	x		Disparity error counter for FltxDispFc. Counts up when disparity error is detected on lane. Wraps on maximum count of 8'hFF if Wrap is set. Note: Counter does not count up in clock cycle in which FltxDispFc register is being read or written to by SCB.

2.16.6 R_FltxAltNull

Register

R_FltxAltNull

Address

0x0_0000_0014 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
4	Ena	RW	1		Enable driving AltNull during IDLE cycle. When clear, AltNull will not be driven at any setting of ANullRate.
3:0	Rate	RW	8		Rate of AltNull during IDLE cycles. When ANullEnable and setting is 0, only altNull will be driven on IDLE cycles.

2.16.7 R_FltxHeartbeat

Register

R_FltxHeartbeat

Attributes

-writeonemixed -kernel

Address

0x0_0000_0018 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
13	Intr	RW1C	0		Heartbeat error interrupt from Fltx. This bit is set if IntEna is set AND loss of heartbeat occurs in MissionMode. All 3 ways of losing MissionMode (force-retraining, loss-of-link-health, and heartbeat-timeout) are considered a Heartbeat Error, and will cause a Heartbeat error interrupt. Once Intr bit is set, it will need to be cleared or disabled to clear the main interrupt from the link.
12	IntEna	RW	0		Heartbeat error interrupt enable for Fltx.
11	Init	RWS	0		Heartbeat Init. For every transition of 0-to-1, heartbeat counter is initialized to its reset state once. Note: Writing 1 to this field has side effect.
10	Dis	RW	0		Heartbeat Disable. When set, heartbeat never expires and thus heartbeat function is disabled.
9:0	Threshold	RW	0x080		Heartbeat Threshold. Holds threshold value in max number of clock cycles during which heartbeat must be detected.

2.16.8 R_FltxDriveError

Register

R_FltxDriveError

Address

0x0_0000_001c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
23:16	TBadChar	RWS	0		Drive Bad Character. On transition from 0-to-1, bad character is driven on lane. Each lane is assigned a bit in this field.
15:8	TBadDisp	RWS	0		Drive Bad disparity. On transition from 0-to-1, bad disparity is driven on lane. Each lane is assigned a bit in this field.
7:0	TCharError	RW	0		Create transmit Error. A bit is assigned to each lane. A bit is set to reflect created error on lane as per either TBadChar or TBadDisp field. In system level testing, error created on lane(s) should also be detected by corresponding 8B10B decoder lanes of receiver chip.

2.16.9 R_FltxTxLcStatus

Register

R_FltxTxLcStatus

Attributes

-noregtestcpu -kernel

Address

0x0_0000_0020 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
8	AllReset	R	x		Holds status of flag-AllLanesReset. Note that this status field holds status of one receiver lane. When set, it indicates that PLL is locked and lane has its reset de-asserted.
7	Linkhealth	R	x		Holds status of flag-LinkHealth.
6	TxLinkSync	R	x		Holds status of flag-TxLinkSync.
5	MissionMode	R	x		Holds status of flag-MissionMode.
4	Heartbeat	R	x		Holds status of flag-Heartbeat.
3:0	Steps	R	x		Holds status of Step-1,2,3,4 of TxLC.

2.16.10 R_FltxTxLcControl

Register

R_FltxTxLcControl

Attributes

-kernel

Address

0x0_0000_0024 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
1	Ena	RWS	0		Enable TxLinkSync. When set, hardware execution routine TxLinkSync is enabled. After setting this bit, write ForceRT bit to initiate TxLinkSync.
0	ForceRT	RWS	0		Force Retraining or execute TxLinkSync routine. ON transition from 0-to-1 of this bit will force re-entry to TxLinkSync routine.

2.16.11 R_FltxTxLcCount

Register

R_FltxTxLcCount

Attributes

-kernel

Address

0x0_0000_0028 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
7:0	TxLcCount	RW	0		TcLcCount. Counter holding number of times hardware routine Txlc is evoked. The counter will count up when TxLc goes from Step-1 to Step-2. Counter will wrap on maximum count.

2.16.12 R_FltxS2WaitTime

Register

R_FltxS2WaitTime

Address

0x0_0000_002c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
9:0	Step2WaitTime	RW	0x7F		Step2 sleep timer value. Cycles to wait in step2. Default value is set at 7F(hex) i.e 127 x 5 = 635ns.

What is Step2WaitTime?

The Step2WaitTime is the time required to insure that Link between two ICE9 is filled with NULL characters only. The default setting of 0x7f is initialized at power-on which equals the waiting time of 635ns in system when SCLK is operating at 200 MHz. To change Step2WaitTime setting after power-on, (a) put FLT into SoftReset, then (b) write the new value into S2WaitTime, and then (c) remove SoftReset. Also it is strongly suggested to avoid depositing any value lower than 0x0f as Step2WaitTime because such lower value may not be sufficient to insure that Link between two ICE9 is filled with NULL characters.

2.16.13 Fltx Manual Override Rotator (MOR)

Register

R_FltxMOR

Address

0x0_0000_0030 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
4	ManualOverrideRotator	RW	0		Manual override or Force Rotator Setting for flow control lane. When set, rotator function in framer is disabled and rotator value specified in RotatorSetting is forced.
3:0	RotatorSetting	RW	0		Rotator Setting. Note that Rotator setting from 0x9 to 0xF are assumed to be at value of 0x9.

How Manual Rotator Override function works?

Manual Override Rotator (MOR) function may be activated if automatic Linksync routine fails and failure points to rotator function.

1. To activate MOR, select RotatorSetting (between 0x0 through 0x9) and set ManualOverrideRotator bit.

2. Next, initiate Linksync routine by accessing R-FltxTxLcControl register.

3. During discovery of valid RotatorSetting, Rotator field of R_FltxLaneStatus is invalid because it captures rotator setting in automatic Linksync routine. However, Lanehealth field of R_FltxLaneStatus will indicate valid status of lane health. In MOR, if LaneHealth is true then correct rotator setting has been acquired other wise other values of rotator setting should be tried.

4. During MOR, all fields of R_FltxTxLcStatus are valid and content of this register should be used to find correct rotator setting.

5. After discovering correct rotator setting re-initiate Linksync routine by accessing R-FltxTxLcControl register.

2.16.14 R_FltxFarEndLoopback

Register

R_FltxFarEndLoopback

Address

0x0_0000_0034 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
0	FarEndLpBk	RW	0		Far End Loopback Mode. When set, it indicates that Far end loopback mode is active. When set, puts both, the FLT and FLR, with the same link number into FarEndLoopback mode as described in section-2.11.2.

2.16.15 R_FltxBBDiag

Register

R_FltxBBDiag

Address

0x0_0000_0040 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
15	BBEnab	RW	0		Bit Blasting mode enable.
14:12	FcPattern	RW	0		Receive bit-blasting pattern type on flow control lane. (a) 0x0 - repeat k28.5 (PNULL) 31 times and k28.0 (ANULL) (once) (b) 0x1 - PNULL (k28.5) (c) 0x2 - D10.2 (0x4A) (d) 0x3 - D24.3 (0x78) (e) 0x4 - IKJPAT pattern to stimulate inter-symbol interference (ISI) in ac-coupled system. Loop of 484 Character: ..... D30.3 (0x7E) 167 times ..... D20.3 (0x74) once ..... D30.3 (0x7E) once ..... D11.5 (0xAB) once ..... D21.5 (0xB5) 51 times ..... D30.2 (0x5E) once ..... D10.2 (0x4A) once ..... D30.3 (0x7E) 4times ..... D30.7 (0xFE) once .....D20.7, D11.7 (0xF4EB) 128 times
11	FcLaneSel	RW	0		Flow control lane select for bit-blasting pattern. When set, flowcontrol lane is receiving pattern selected by FcPattern field.
10:8	TxPattern	RW	0		Transmitter bit-blasting pattern type. (a) 0x0 - repeat driving k28.5 (PNULL) 31 times and k28.0 (ANULL) (once) (b) 0x1 - drive PNULL (k28.5) (c) 0x2 - drive D10.2 (0x4A) (d) 0x3 - drive D24.3 (0x78) (e) 0x4 - drive IKJPAT pattern to stimulate inter-symbol interference (ISI) in ac-coupled system. Loop of 484 Character: ..... D30.3 (0x7E) 167 times ..... D20.3 (0x74) once ..... D30.3 (0x7E) once ..... D11.5 (0xAB) once ..... D21.5 (0xB5) 51 times ..... D30.2 (0x5E) once ..... D10.2 (0x4A) once ..... D30.3 (0x7E) 4times ..... D30.7 (0xFE) once .....D20.7, D11.7 (0xF4EB) 128 times
7:0	TxLaneSel	RW	0		Transmitter lane select for bit-blasting pattern. A bit is assigned to each of 8 lanes. When BBEnable and this is set, selected lane is enabled to drive bit-blasting pattern as selected by TxPattern field. When BBEnable and this bit is clear, selected lane drives PNULL (k28.5) pattern.

2.16.16 Fltx_BBDiagStatus

Register

R_FltxBBDiagStatus

Attributes

-noregtestcpu

Address

0x0_0000_0044 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
1	FcLaneSync	R	x		Lane synchronization status of flow control lane. This bit will be set in BBMode, when dlow control lane is selected to check for FcbbPattern and it finds FcbbPattern. This bit will remain clear in BBMode, if flow control LaneSelect bit is clear.
0	FcBBError	R	x		Bit Blasting error on flow control lane. This bit will be set in BBMode, if LaneSync is set and thenflow control lane detects BBPattern error. Otherwise this bit will remain clear. This bit will also remain clear if flow control LaneSelect bit is clear.

2.17 FLR Registers

2.17.1 R_FlrxSoftReset

Register

R_FlrxSoftReset

Attributes

-kernel

Address

0x0_0000_0000 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
0	SoftReset	RW	0		Reset Link when set. When written 1, receiver link remains in reset state. When written 0, the receiver link logic come out of the reset state. FlrCsr module remains unaffected by SoftReset.

Operation of SoftReset

When SoftReset is asserted, all CSRs of FLRx remain unaffected by SoftReset. However, control flops within FLRx module are intialized to power-on reset value. After de-assertion of SoftReset, software will have to initiate skipbeat function on its receiver lanes and then enable receiver link.

2.17.2 R_FlrxLinkStatus

Register

R_FlrxLinkStatus

Attributes

-noregtestcpu -kernel

Address

0x0_0000_0004 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
9	DrivenBadFcChar	RW	0		Drove bad FC character and created transmit Error. When set, it reflects that transmit error was created by setting either DriveBadChar or DriveBadDisp of R_FLrxLinkControl. In system level testing, error created on flow control lane should also be detected by corresponding 8B10B decoder lane of receiver chip.
8	PllLock	R	x		Lock status of TxPLL of FC lane.
7:0	CdrPllLock	R	x		Lock status of CdrPLL. Holds lock status of CDR PLL of eight receiver PLLs in QPMA.

2.17.3 R_FlrxLinkControl

Register

R_FlrxLinkControl

Address

0x0_0000_0008 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
1	DriveBadChar	RWS	0		Drive bad character. On transition from 0-to-1, one bad or invalid character is driven on FC lane.
0	DriveBadDisp	RWS	0		Drive bad disparity. On transition from 0-to-1, one character is driven with disparity error on FC lane.

2.17.4 R_FlrxRotator

Register

R_FlrxRotator

Address

0x0_0000_000c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Rotator	R	x		Rotator Status. Rotator status of eight lanes. Each lane is assigned 4-bit wide field.

2.17.5 R_FlrxRxLcStatus

Register

R_FlrxRxLcStatus

Attributes

-noregtestcpu -kernel

Address

0x0_0000_0010 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
8	AllRxLanesReset	R	x		Holds status of flag-AllRxLanesReset. When set, it indicates that PLL is locked and eight lanes have their reset signals de-asserted.
7	Linkhealth	R	x		Holds status of flag-LinkHealth.
6	RxLinkSync	R	x		Holds status of flag-RxLinkSync.
5	MissionMode	R	x		Holds status of flag-MissionMode.
4	Heartbeat	R	x		Holds status of flag-Heartbeat.
3:0	Steps	R	x		Holds status of Step-1,2,3,4 of RxLC.

2.17.6 R_FlrxLaneHealth

Register

R_FlrxLaneHealth

Attributes

-kernel

Address

0x0_0000_0014 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
15:8	ClrLaneHealth	RW	0		Clear lane health. On transition from 0-to1, lane's health bit is cleared. Each lane is assigned a bit in this field.
7:0	LaneHealth	R	x		Lane health status. Each lane is assigned 1-bit field.

2.17.7 R_FlrxWSyncMode

Register

R_FlrxWSyncMode

Address

0x0_0000_0018 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
27:20	MwsEnab	RW	0		Manual WordSync Enable. This field is 8-bit wide and one bit is assigned to each lane. When Mws_Enab[x] is set, corresponding 2-bit wide lane selector setting of wsync multiplexer is forced. When MwsEnab[x] is clear, corresponding wsync multiplexer setect setting is set by automatic wordsync operation.
19:4	Mws	RW	0		Manual Wordsync setting. This field is 16-bit wide and has 8 groups. Each group is 2-bit wide and assigned to a lane. Bits[5:4] are assigned to Lane-0, bit[7:6] are assigned to Lane-1, and so on. For each lane, 2-bit field holds select value for 4-to-1 wsync multiplexer.
3:2					Reserved.
1	ForceWsync	RW	0		Force Wsync cycle. On transition from 0-to1 of this bit forces RxLinkSync routine to enter in Step4.
0	DisVerror	RW	0		Disable Wsync pattern verification error. If this bit is set then pattern verification logic in step-4 does not detect any errors. Setting of this bit allows successful completion of step-4.

How Manual Override Wordsync mode works?

Manual Wordsync operation can be invoked if skipbeat and rotator functions are working but unexplained crc errors (without disparity error(s) and/or invalid character error(s)) are observed. In Manual Wordsync Override mode, link is forced to enter MissionMode so that characters from all 8 lanes are sent to fabric switch. Though each lane may be individually forced in Manual Override Wordsync mode (MwsEnab), preferred method is to force all 8 lanes in Manual Override Wordsync mode be setting all bits of MwsEnab field and by selecting individual lane's 2-bit wordsync setting (Mws).

When MwsEnab is set, once link enters Step4, it stays in Step4 for the duration of time it takes to execute tasks of step4 (approximately 4 microsec) and then forces link to enter MissionMode. After link has entered MissionMode, R_FlrxWsyncStatus_AutoSetting field is invalid but the rest of the bits of R_FlrxWsyncStatus are valid.

2.17.8 R_FlrxWSyncStatus

Register

R_FlrxWSyncStatus

Address

0x0_0000_001c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
19:4	AutoSetting	R	x		Wsync Auto Setting. This field is 16-bit wide and has 8 groups. Each group is 2-bit wide and assigned to each lane. Lane-0 has bits[1:0], Lane-1 has [3:2], and so on. Reads wsync multiplexer settings in wsync auto operation. Reading of this field is invalid if corresponding lane's Mws_Enab bit is set in R_FlrxWSyncMode register.
3	Status	R	x		Status of wordsync operation. Set to 1 when wordsync is successful.
2	VError	R	x		This status bit is set when verify cycle detects error during Wsync.
1	Seek	R	x		This status bit is set when wsync cycle is active.
0	Verify	R	x		This status bit is set when verify cycle is active during Wsync .

2.17.9 R_FlrxHeartbeat

Register

R_FlrxHeartbeat

Attributes

-writeonemixed -kernel

Address

0x0_0000_0020 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
13	Intr	RW1C	0		Heartbeat error interrupt from Flrx. This bit is set if IntEna is set AND loss of heartbeat occurs in MissionMode. All 3 ways of losing MissionMode (force-retraining, loss-of-link-health, and heartbeat-timeout) are considered a Heartbeat Error, and will cause a Heartbeat error interrupt. Once Intr bit is set, it will need to be cleared or disabled to clear the main interrupt from the link.
12	IntEna	RW	0		Heartbeat error interrupt enable for Flrx.
11	Init	RWS	0		Heartbeat Init. For every transition of 0-to-1, heartbeat counter is initialized to its reset state once. Note: Writing 1 to this field has side effect.
10	Dis	RW	0		Heartbeat Disable. When set, heartbeat never expires and thus heartbeat function is disabled.
9:0	Threshold	RW	128		Heartbeat Threshold. Holds threshold value in max number of clock cycles during which heartbeat must be detected.

2.17.10 R_FlrxRxLcControl

Register

R_FlrxRxLcControl

Attributes

-kernel

Address

0x0_0000_0024 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
1	Ena	RWS	0		Enable RxLinkSync. When set, hardware execution routine RxLinkSync is enabled. After setting this bit, write ForceRT bit to initiate RxLinkSync.
0	ForceRT	RWS	0		Force Retraining or execute RxLinkSync routine. Setting of this bit will force re-entry to RxLinkSync routine.

2.17.11 R_FlrxRxLcCount

Register

R_FlrxRxLcCount

Attributes

-kernel

Address

0x0_0000_0028 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
7:0	RxLcCount	RW	0		RxLcCount. Counter hoilding number of times hardware routine Rxlc is evoked. The counter will count up when RxLc goes from Step-1 to Step-2. Counter will wrap on maximum count.

2.17.12 R_FlrxS2WaitTime

Register

R_FlrxS2WaitTime

Address

0x0_0000_002c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
9:0	Step2WaitTime	RW	0x7F		Step2 sleep timer value. Cycles to wait in step2. Default value is set at 7F(hex) i.e 127 x 5 = 635ns.

What is Step2WaitTime?

The Step2WaitTime is the time required to insure that Link between two ICE9 is filled with NULL characters only. The default setting of 0x7f is initialized at power-on which equals the waiting time of 635ns in system when SCLK is operating at 200 MHz. To change Step2WaitTime setting after power-on, (a) put FLR into SoftReset, then (b) write the new value into S2WaitTime, and then (c) remove SoftReset. Also it is strongly suggested to avoid depositing any value lower than 0x0f as Step2WaitTime because such lower value may not be sufficient to insure that Link between two ICE9 is filled with NULL characters.

2.17.13 Flrx Lane Invalid Character Error Register

Register

R_FlrxLaneInvC[7:0]

Attributes

-kernel -writeonemixed

Address

0x0_0000_0030 - 0x0_0000_004c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
18	Intr	RW1C	0		Invalid Character error interrupt from FlrxLaneInvC. This bit is set if IntEna is set AND (Compare == Counter).
17	IntEna	RW	0		Invalid Character error interrupt enable for FlrxLaneInvC.
16	Wrap	RW	0		Enable wrap mode for FlrxLaneInvC. When set, Counter wraps on maximum count.
15:8	Compare	RW	0		Invalid character error counter comparator for FlrxLaneInvC.
7:0	Counter	RW	x		Invalid character error counter for FlrxLaneInvC. Counts up when invalid charater error is detected on lane. Wraps on maximum count of 8'hFF if Wrap is set. Note: Counter does not count up when FlrxLaneInvC register is being read or written to by SCB.

2.17.14 Flrx Lane Disparity Error Register

Register

R_FlrxLaneDisp[7:0]

Attributes

-kernel -writeonemixed

Address

0x0_0000_0050 - 0x0_0000_006c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
18	Intr	RW1C	0		Disparity error interrupt from FlrxLaneDisp. This bit is set if IntEna is set AND (Compare == Counter).
17	IntEna	RW	0		Disparity error interrupt enable for FlrxLaneDisp.
16	Wrap	RW	0		Enable wrap mode for FlrxLaneDisp. When set, Counter wraps on maximum count.
15:8	Compare	RW	0		Disparity error counter comparator for FlrxLaneDisp.
7:0	Counter	RW	x		Disparity error counter for FlrxLaneDisp. Counts up when disparity error is detected on lane. Wraps on maximum count of 8'hFF if Wrap is set. Note: Counter does not count up when FlrxLaneDisp register is being read or written to by SCB.

2.17.15 R_Flrx Lane Status Register

Register

R_FlrxLaneStatus[7:0]

Attributes

-noregtestcpu -kernel

Address

0x0_0000_0070 - 0x0_0000_008c (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
6	SbTestaRxClkN	R	x		Test aRxClkN signal. It holds sampled value of aRxClkN signal from Qpma.
5	SbSuccess	R	x		SkipBeat Success. It indicates status of last skipbeat operation. When set, indicates that SkipBeat function has been successful.
4	SbActive	R	x		SkipBeat Active. When set, indicates that SkipBeat operation is active.
3	SbFirstSearch	R	x		State of SkipBeat First search function. When set, indicates that the First Search is completed.
2	SbSecondSearch	R	x		State of SkipBeat Second search function. When set, indicates that Second search is completed.
1	SbFinalSearch	R	x		State of SkipBeat Final search function. When set, indicates that Final search is completed.
0	SbAdjust	R	x		State of SkipBeat Adjust function. When set, indicates that Adjustment is completed.

2.17.16 Flrx Lane Control Register

Register

R_FlrxLaneControl[7:0]

Attributes

-kernel

Address

0x0_0000_0090 - 0x0_0000_00ac (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
7	ForceSkipBeat	RW	0		Force Skipbeat. This bit must remain clear when SkipBeatEnable is clear. When SkipBeatEnable is set : For every transition of 0-to-1 of this bit, RxClk offset is skipped 1-bit time. This field is intended to be used in manual setting of RxClk. This bit should be clear after manual setting of RxClk is completed.
6:5					Reserved.
4	SkipBeatEnable	RW	0		Skip Beat Enable. At the transition from 0-to-1, SkipBeat function is executed once using value selected in “SkipBeatOffset”. To initialize Skip Beat function, write 1 followed by write 0. For manual setting of skipbeat, write 1, then use ForceSkipBeat (above), then write this bit 0.
3:0	SkipBeatOffset	RW	5		SkipBeat Offset. The receiver RxClk offset is equal to “SkipBeatOffset” bit-time wrt sclk. The power-on default value is 5(hex). This field is 4-bit wide and SkipBeatOffset can be selected from 0(hex) to 9(hex). The values in this field are modulo-10. For applying newer value of SkipBeatOffset, SkipBeatEnable should be toggled.

Operating modes of Skipbeat function

At the end of reset sequence, SkipBeatOffset field value defaults to 0x5. It holds offset value in bit-time. At 200Mhz of sclk, bit time is 0.5nsec.

SCB master can modify SkipBeatOffset value and invoke skipbeat function by toggling SkipBeatEnable bit once. This method triggers skipbeat function with selected SkipBeatOffset value. Please note that during this process, if any time reset sequence is invoked then SkipBeatOffset will be defaulted to 0x5.

For manual SkipBeat setting, set SkipBeatEnable=1, (the SkipBeat function will run, completing faster than you can do your next register access), then use single step (sample and move) manual skipbeat algorithm. To do this, repeatedly (a) sample state of “SbTestaRxClkN” of R_FlrxLaneStatus register, and (2) move phase of receiver clock 1-bit time by toggling “ForceSkipBeat”. Note where SbTestaRxClkN transitions 0-to-1 and 1-to-0, then do additional skips to position the offset correctly relative to those transitions. After desired phase alignment of receiver clock is achived, SkipBeatEnable bit should be cleared.

2.17.17 Flrx Manual Override Rotator (MOR)

Register

R_FlrxMOR[7:0]

Address

0x0_0000_00b0 - 0x0_0000_00cc (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
4	ManualOverrideRotator	RW	0		Manual override or Force Rotator Setting. When set, rotator function in framer is disabled and rotator value specified in RotatorSetting is forced.
3:0	RotatorSetting	RW	0		Rotator Setting. Note that Rotator setting from 0x9 to 0xF are assumed to be at value of 0x9.

How Manual Override Rotator function works?

Manual Override Rotator (MOR) function may be activated if automatic Linksync routine fails and failure points to rotator function.

1. To activate MOR on failing lane, select RotatorSetting (between 0x0 through 0x9) and set ManualOverrideRotator bit.

2. Next, initiate Linksync routine by accessing R-FltxRxLcControl register.

3. During discovery of valid RotatorSetting, values in R_FlrxRotator is invalid because it captures rotator setting in automatic Linksync routine. However, Lanehealth field of R_FlrxRxLcStatus will indicate valid status of lane health. In MOR, if LaneHealth is true then correct rotator setting has been acquired other wise other values of rotator setting should be tried.

4. During MOR, all fields of R_FlrxRxLcStatus are valid and content of this register should be used to find correct rotator setting.

5. After discovering correct rotator setting re-initiate Linksync routine by accessing R-FlrxRxLcControl register.

2.17.18 R_FlrxBBDiag

Register

R_FlrxBBDiag

Address

0x0_0000_00d0 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
15	BBEnab	RW	0		Bit Blasting mode enable.
14:12	FcPattern	RW	0		Flow control bit-blasting pattern type. (a) 0x0 - repeat driving k28.5 (PNULL) 31 times and k28.0 (ANULL) (once) (b) 0x1 - drive PNULL (k28.5) (c) 0x2 - drive D10.2 (0x4A) (d) 0x4 - drive D24.3 (0x78) (e) 0x8 - drive IKJPAT pattern to stimulate inter-symbol interference (ISI) in ac-coupled system. Loop of 484 Character: ..... D30.3 (0x7E) 167 times ..... D20.3 (0x74) once ..... D30.3 (0x7E) once ..... D11.5 (0xAB) once ..... D21.5 (0xB5) 51 times ..... D30.2 (0x5E) once ..... D10.2 (0x4A) once ..... D30.3 (0x7E) 4times ..... D30.7 (0xFE) once .....D20.7, D11.7 (0xF4EB) 128 times
11	FcLaneSel	RW	0		Flow control lane select for bit-blasting pattern. When BBEnab and this bit is set, flowcontrol lane transmits pattern selected by FcPattern field. When BBEnable and this bit is clear, flow control lane transmits PNULL (k28.5) pattern.
10:8	RxPattern	RW	0		Receiver bit-blasting pattern type. (a) 0x0 - repeat k28.5 (PNULL) 31 times and k28.0 (ANULL) (once) (b) 0x1 - PNULL (k28.5) (c) 0x2 - D10.2 (0x4A) (d) 0x3 - D24.3 (0x78) (e) 0x4 - IKJPAT pattern to stimulate inter-symbol interference (ISI) in ac-coupled system. Loop of 484 Character: ..... D30.3 (0x7E) 167 times ..... D20.3 (0x74) once ..... D30.3 (0x7E) once ..... D11.5 (0xAB) once ..... D21.5 (0xB5) 51 times ..... D30.2 (0x5E) once ..... D10.2 (0x4A) once ..... D30.3 (0x7E) 4times ..... D30.7 (0xFE) once .....D20.7, D11.7 (0xF4EB) 128 times
7:0	RxLaneSel	RW	0		Receiver Lane Select for bit-blasting patterns. A bit is assigned to each of 8 lanes. When set, selected lane is enabled to check bit-blasting pattern type selected by RxPattern field. When clear, selected lane does not check for RxbbPattern.

2.17.19 Flrx_BBDiagStatus

Register

R_FlrxBBDiagStatus

Attributes

-noregtestcpu

Address

0x0_0000_00d4 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
15:8	RxLaneSync	R	x		Lane synchronization status. A bit is assigned to each lane. This bit will be set in BBMode, when corresponding lane is selected to check for RxbbPattern and selected lane finds RxbbPattern. This bit will remain clear in BBMode, if corresponding lane is not selected.
7:0	RxBBError	R	x		Bit Blasting error. A bit is assigned to each lane. This bit will be set in BBMode, if corresponding LaneSync is set and then if selected lane detects BBPattern error. Otherwise this bit will remain clear. This bit will also remain clear if corresponding LaneSelect bit is clear.

2.18 FLR/FLT Register Allocation

This chapter instantiates the three copies of the FLR and FLT registers.

2.18.1 Flr0

Register

R_Flr0* : R_Flrx*

Address

0xE_0D00_0000-0xE_0DFF_FFFF

2.18.2 Flr1

Register

R_Flr1* : R_Flrx*

Address

0xE_1D00_0000-0xE_1DFF_FFFF

2.18.3 Flr2

Register

R_Flr2* : R_Flrx*

Address

0xE_2D00_0000-0xE_2DFF_FFFF

2.18.4 Flt0

Register

R_Flt0* : R_Fltx*

Address

0xE_3D00_0000-0xE_3DFF_FFFF

2.18.5 Flt1

Register

R_Flt1* : R_Fltx*

Address

0xE_4D00_0000-0xE_4DFF_FFFF

2.18.6 Flt2

Register

R_Flt2* : R_Fltx*

Address

0xE_5D00_0000-0xE_5DFF_FFFF

Vregs_End_Of_Decl

2.19 Quad Serdes Physical Media Access (QPMA)

The AnalogBits SERDES physical media access macro, referred to as QPMA, has quad transmit and receive lanes. The transmit and receive lanes within QPMA are identified as X, Y, Z, and W. The QPMA has quad clock and data recovery (CDR) logic for quad receiver lanes. The QPMA generates four seperate receiver clocks, one for each receiver lane. The receiver clock is in phase with incoming data streams. The QPMA has one PLL which generates clocks for four transmit lanes. The QPMA also has the calibration and impedance control circuits for quad transmitter and receiver channels.

Each ICE9 has 3 fabric links and each fabric link has 9 serdes lanes. Hence each ICE9 will use 7 QPMAs to constuct 3 fabric links. The Figure-2.9 shows the placement of 7 QPMA in ICE9. They are numbered from 0 through 6. The QPMA6 supports flow control lanes for each of the three links. The QPMA6 will have one pair of unused transmit and receive lane.

Following table shows the lane assignemnts in QPMA for each of the three fabric links.

Fabric Link	Link Lane	QPMA	QPMA Lane	Note
FLT0/1/2	0	0/2/4	T_Z	
	1	0/2/4	T_Y	
	2	0/2/4	T_X	
	3	0/2/4	T_W	
	4	1/3/5	T_Z	
	5	1/3/5	T_Y	
	6	1/3/5	T_X	
	7	1/3/5	T_W	
FLT0	FC	6	R_Z	
FLT1	FC	6	R_Y	
FLT2	FC	6	R_W	
		6	R_X	Unused

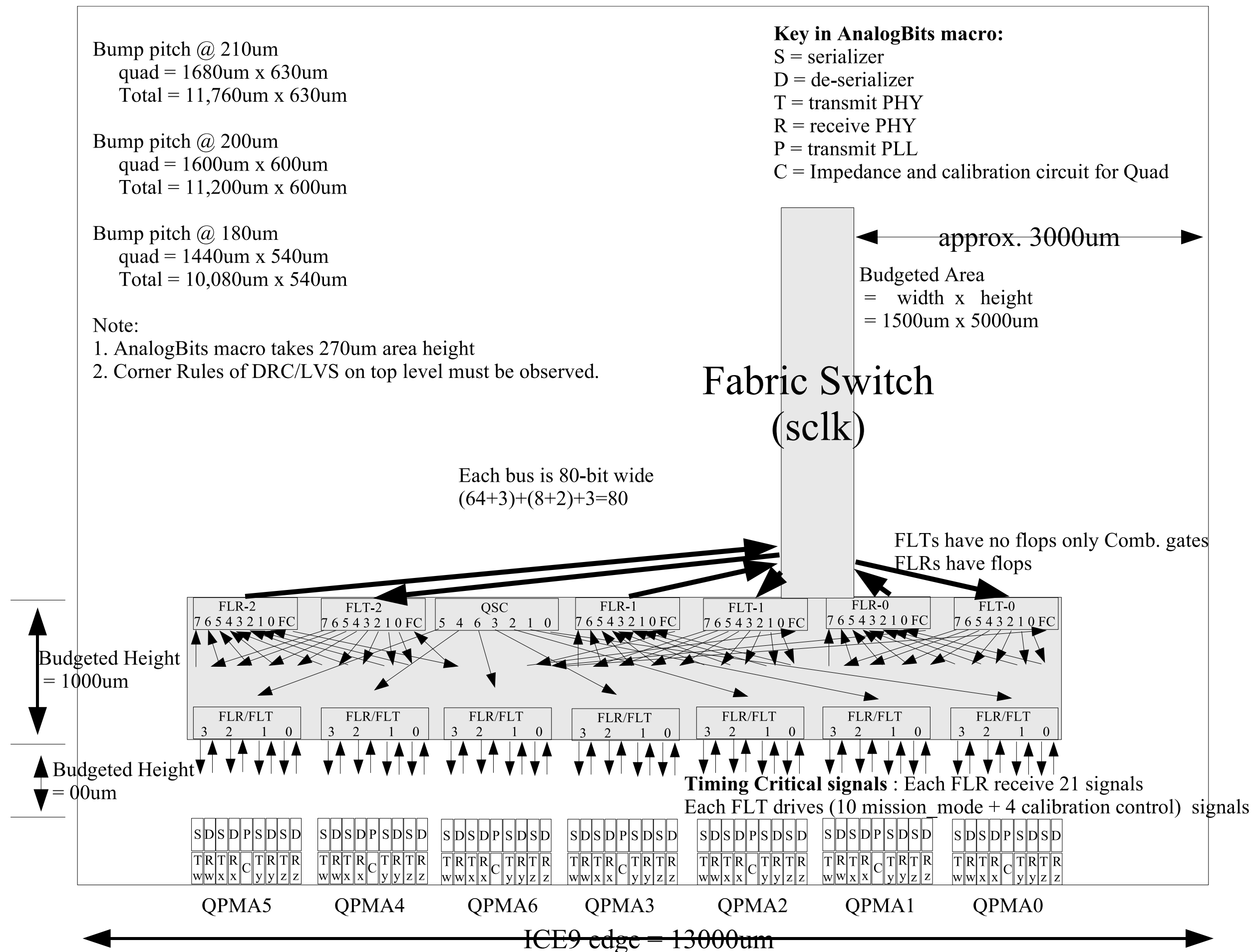


Figure 2.9: QPMA Placement in ICE9

Fabric Link	Link Lane	QPMA	QPMA Lane	Note
FLR0/1/2	0	0/2/4	R_Z	
	1	0/2/4	R_Y	
	2	0/2/4	R_X	
	3	0/2/4	R_W	
	4	1/3/5	R_Z	
	5	1/3/5	R_Y	
	6	1/3/5	R_X	
	7	1/3/5	R_W	
FLR0	FC	6	T_Z	
FLR1	FC	6	T_Y	
FLR2	FC	6	T_W	
		6	T_X	Unused

2.19.1 Calibration and Impedance Control of the driver and Receiver

The QPMA has individual transmitter driver impedance control circuitry. The QPMA also has individual receiver impedance calibration circuitry. The details of the driver and receiver control is described in AnalogBit's document "Serdes PMA Programmer's Reference Manual".

ICE9 has the driver and receiver handshake interface with QPMA which is called the Quad Serdes Control (QSC). The QSC has 5 registers which are accessible through SCB. Those 5 registers are QscGo, QscCA, QscSerDatAR, QscSerDatT, and QscSerDatP. The QscCA holds the address of the target driver or receiver. The

QscSerDat* registers hold calibration values for targeted driver and receiver. By writing 1 to QscGo register, the QSC will load impedance and calibration values in target driver or receiver. The QSC also has the QscStatus register which holds the status of the handshake as described in section-2.20.

2.19.2 Verification Checklist:

1. Reset sequence
2. Transmitter channel impedance calibration
3. Receiver channel impedance calibration

2.20 Quad Serdes Control (QSC) Registers

2.20.1 R_QscGo

Register

R_QscGo

Attributes

-kernel

Address

0xE_6D00_0000

Bit	Mnemonic	Access	Reset	Type	Definition
0	QscGo	RWS	0		Write QSC register for specified QuadSerdes. On the transition of 0-to-1, targeted QuadSerdes register is written. The target of the QuadSerdes Register is specified by R_QscCA register and the data values are specified in R_QscSerDatAR, R_QscSerDatT, and R_QscSerDatP registers.

2.20.2 R_QscStatus

Register

R_QscStatus

Attributes

-kernel

Address

0xE_6D00_0004

Bit	Mnemonic	Access	Reset	Type	Definition
6	InvalidQadr	RW1C	0		Invalid QuadSerdes address. This bit is set if QSC was invoked with atleast one invalid Quad Serdes address since previous clearing of this field.
5	InvalidSubQadr	RW1C	0		Invalid Sub Quad Address. This bit is set if QSC was invoked with atleast one invalid Sub Quad address since previous clearing of this field.
4	InvalidTarget	RW1C	0		Invalid Target. This bit is set if QSC was invoked with atleast one invalid target address since previous clearing of this field.
3:2					Reserved.
1	QscSuccess	R	0		QSC Success. This bit holds status of the previous QSC transaction. When set, it indicates that the previous QSC transaction was a success. When clear, it indicates that the previous QSC transaction was a failure.
0	Busy	R	0		QSC busy. Holds status of QSC controller. When set, it indicates that QSC controller is busy.

2.20.3 R_QscCA

Register

R_QscCA

Attributes

-kernel

Address

0xE_6D00_0010

Bit	Mnemonic	Access	Reset	Type	Definition
11:8	QscAdr	RW	0		QSC Address. Holds address of QSC. There are total of 7 QSC. If this field has value greater than 6(hex) then it makes invalid QSC address.
7:4	QscSubAdr	RW	0		QSC Sub Address. Holds sub address of QSC. There are total of 4 subaddresses in each QSC. The encodings of this field is as below: 8(hex) - Sub address W 4(hex) - Sub address X 2(hex) - Sub address Y 1(hex) - Sub address Z All other encodings (total of 12 of them) makes invalid QSC sub address.
3:0	QscTarget	RW	0		QSC Target. Holds target of the calibration transaction. 1(hex) - Tx Driver. 2(hex) - Rx Receiver All other encodings (total of 14 of them) makes invalid target.

2.20.4 R_QscSerDatAR

Register

R_QscSerDatAR

Address

0xE_6D00_0014

Bit	Mnemonic	Access	Reset	Type	Definition
17:0	ASerDatAR	RW	0		aSerDatAR Register. Holds 18-bit value tobe written in either aSerDatA of aSerDatR register of QSC. Refer to (a) section-7.3 of Serdes Programmer's Reference Manual for Transmitter ouput driver settings and (b) section-8.3 of Serdes Programmer's Reference Manual for Receiver settings.

2.20.5 R_QscSerDatT

Register

R_QscSerDatT

Address

0xE_6D00_0018

Bit	Mnemonic	Access	Reset	Type	Definition
17:0	ASerDatT	RW	0		aSerDatT Register. Holds 18-bit value tobe written in aSerDatT register of QSC. Refer to section-7.3 of Serdes Programmer's Reference Manual for Transmitter ouput driver settings.

2.20.6 R_QscSerDatP

Register

R_QscSerDatP

Address

0xE_6D00_001c

Bit	Mnemonic	Access	Reset	Type	Definition
17:0	ASerDatP	RW	0		aSerDatP Register. Holds 18-bit value tobe written in aSerDatP register of QSC. Refer to section-7.3 of Serdes Programmer's Reference Manual for Transmitter ouput driver settings.

2.20.7 R_QscQpmaStatus

Register

R_QscQpmaStatus[6:0]

Attributes

-noregtestcpu -kernel

Address

0xE_6D00_0020 - 0xE_6D00_0038

Bit	Mnemonic	Access	Reset	Type	Definition
16:13	PllBpStatus	R	x		PLL Bypass Test Status. When set, indicates that PLL Bypass Test was successful for [W,X,Y,Z] lanes.
12	ZCompOp	R	x		Impedance calibrator result. When 1, Z < nominal. When 0, Z > nominal.
11:8	CdrDiagOut	R	x		CDRDiagOut. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. Refer to “Serdes PMA Programmer’s Reference Manual” for detailed explanation.
7	RefClkStable	R	x		RefClk (or sclk) stable. When set, indicates that sclk is stable. This signal is generated by CLK_GEN.
6					Reserved.
5	TxClkStable	R	x		Set 1024 sclk cycles after ATxClkStable is asserted. When set, it indicates that TxClk is stable.
4	ATxClkStable	R	x		Set when transmitter clocks are up and stable for [W,X,Y,Z] lanes.
3	ARxClkStableW	R	x		Set when receiver W-lane clock is bit-locked to incoming data stream.
2	ARxClkStableX	R	x		Set when receiver X-lane clock is bit-locked to incoming data stream.
1	ARxClkStableY	R	x		Set when receiver Y-lane clock is bit-locked to incoming data stream.
0	ARxClkStableZ	R	x		Set when receiver Z-lane clock is bit-locked to incoming data stream.

2.20.8 R_QscQpmaImpCalibration

Register

R_QscQpmaImpCalibration

Attributes

-kernel

Address

0xE_6D00_003c

Bit	Mnemonic	Access	Reset	Type	Definition
18:12	TxPllRst	RW	0		TxPLL Reset. A bit is assigned to each QPMA. Thus bit-12 controls QPMA0 and bit-18 controls QPMA6. When set, shuts down TxPLL and bypasses RefClk (sclk) to internal high frequency (1 Ghz) clock.
11:10					Reserved.
9:8	ZCalibType	RW	0		Selects which circuitry is calibrated. Asynchronous signal. The encodeds value are as below: 0 - Calibration shutdown 1 - Calib Tx 2 - Calib Rx 3 - invalid
7					Reserved.
6:0	ZCalib	RW	0		Impedance calibration control value. Asynchronous signal.

2.20.9 R_QscQpmaControl

Register

R_QscQpmaControl[6:0]

Attributes

-kernel

Address

0xE_6D00_0040 - 0xE_6D00_0058

Bit	Mnemonic	Access	Reset	Type	Definition
31:28	CDRPLLRst	RW	0		CDRPLL Reset. Asynchronous signal. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. When set, shuts down CDRPLL and bypasses RefClk (sclk) to internal high frequency (1 Ghz) clock.
27:24	RxPwrDown	RW	0		Receiver power down. Asynchronous signal. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. When set, the receiver is in power-down mode. This signal does not include CDRPLL in power-down mode.
23:20	IDDQ	RW	0		IDDQ mode. Asynchronous signal. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. When 1, it is configured for IDDQ mode otherwise the normal operation. Refer to "Serdes PMA Programmer's Reference Manual" for details.
19:16	RxTest	RW	0		RxTest mode control over-ride for CDR feedback loop. Asynchronous signal. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. Refer to "Serdes PMA Programmer's Reference Manual" for details.
15:12	CDRDiagIn	RW	0		CDRDiagIn. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. Refer to "Serdes PMA Programmer's Reference Manual" for details.
11:8	ForceTxHiZ	RW	0		Force driver in HiZ. Asynchronous signal. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. Its assertion takes precedence over SerTxCtr[1:0] load operation.
7:4	ForceRxHiZ	RW	0		Force receiver in HiZ. Asynchronous signal. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. Its assertion takes precedence over SerRxCtr[1:0] load operation.
3:0	LpBkNearEnd	RW	0		When set, NearEndLoopback mode is enabled. Asynchronous signal. Lanes are individually controlled. Lane assignment is [W,X,Y,Z]. When set, a lane is in NearEndLoopback mode. This bit should be 0 for normal mode of operation.

2.20.10 R_QscQpmaTestControl

Register

R_QscQpmaTestControl[6:0]

Address

0xE_6D00_0060 - 0xE_6D00_0078

Bit	Mnemonic	Access	Reset	Type	Definition
15:12	TxHFClkDnB	RW	0xF		Tx HFClk. Asynchronous signal. Lanes are individually controlled. Lane assignement is [W,X,Y,Z]. Refer to “Serdes PMA Programmer’s Reference Manual” for details.
11:8	RxHFClkDnB	RW	0xF		Rx HFClk. Asynchronous signal. Lanes are individually controlled. Lane assignement is [W,X,Y,Z]. Refer to “Serdes PMA Programmer’s Reference Manual” for details.
7:4	RxFDIp1	RW	0		RxFDIp1. Asynchronous signal. Lanes are individually controlled. Lane assignement is [W,X,Y,Z]. Refer to “Serdes PMA Programmer’s Reference Manual” for details.
3:0	RxFDIp0	RW	0		RxFDIp0. Asynchronous signal. Lanes are individually controlled. Lane assignement is [W,X,Y,Z]. Refer to “Serdes PMA Programmer’s Reference Manual” for details.

2.20.11 R_QscInterrupt

Register

R_QscInterrupt

Attributes

-kernel -writeonemixed

Address

0xE_6D00_0080

Bit	Mnemonic	Access	Reset	Type	Definition
7	Intr	RW1C	0		Interrupt signal from Fl. Interrupt signal goes to CSW and it is named fl_csw_Int_sa. This bit is set if IntEnab is set AND any one or more of the FltIntr or FlrIntr bits are set. To clear this bit, first clear all FltIntr and FlrIntr bits that are set, by clearing Intr bits in the registers listed below, then write-1 to this bit.
6	IntEnab	RW	0		Overall interrupt enable from Fl module.
5:3	FltIntr	R	x		Fltx Interrupt status. A bit is assigned to capture interrupt status of each Flt module 2,1, and 0. The FltIntr bit is set for a specific Fltx when one or more of the Intr bits in R_FltxInvCFc, R_FltxDispFc, or R_FltxHeartbeat are set.
2:0	FlrIntr	R	x		Flrx Interrupt status. A bit is assigned to capture interrupt status of each Flr module 2,1, and 0. The FlrIntr bit is set for a specific Flrx when one or more of the Intr bits in R_FlrxHeartbeat, R_FlrxLaneInvC[7:0], or R_FlrxLaneDisp[7:0] are set.

2.20.12 Qsc TxBBDiag

Register

R_QscTxBBDiag

Address

0xE_6D00_0090

Bit	Mnemonic	Access	Reset	Type	Definition
4	BBEnab	RW	0		Bit Blasting mode enable. When reset, 10-bit code sent to this transmitter will remain at logic level low or at value 0x0.
3:1	TxPattern	RW	0		Transmitter bit-blasting pattern type. (a) 0x0 - repeat driving k28.5 (PNULL) 31 times and k28.0 (ANULL) (once) (b) 0x1 - drive PNULL (k28.5) (c) 0x2 - drive D10.2 (0x4A) (d) 0x3 - drive D24.3 (0x78) (e) 0x4 - drive IKJPAT pattern to stimulate inter-symbol interference (ISI) in ac-coupled system. Loop of 484 Character: ..... D30.3 (0x7E) 167 times ..... D20.3 (0x74) once ..... D30.3 (0x7E) once ..... D11.5 (0xAB) once ..... D21.5 (0xB5) 51 times ..... D30.2 (0x5E) once ..... D10.2 (0x4A) once ..... D30.3 (0x7E) 4times ..... D30.7 (0xFE) once .....D20.7, D11.7 (0xF4EB) 128 times
0	TxLaneSel	RW	0		Transmitter lane select for bit-blasting pattern. When BBEnable and this is set, lane is enabled to drive bit-blasting pattern as selected by TxPattern field. When BBEnable and this bit is clear, selected lane drives PNULL (k28.5) pattern.

2.20.13 Qsc Lane Status Register

Register

R_QscLaneStatus

Attributes

-noregtestcpu

Address

0xE_6D00_0094

Bit	Mnemonic	Access	Reset	Type	Definition
15:12	Rotator	R	x		Lane rotator value.
11:10					Reserved.
9	LaneHealth	R	x		Lane health status.
8	CdrPllLock	R	x		CdrPLL lock status. Holds lock status of CDRPLL of unused receiver in QPMA.
7					Reserved.
6	SbTestaRxClkN	R	x		Test aRxClkN signal. It holds sampled value of aRxClkN signal from Qpma.
5	SbSuccess	R	x		SkipBeat Success. It indicates status of last skipbeat operation. When set, indicates that SkipBeat function has been successful.
4	SbActive	R	x		SkipBeat Active. When set, indicates that SkipBeat operation is active.
3	SbFirstSearch	R	x		State of SkipBeat First search function. When set, indicates that the First Search is completed.
2	SbSecondSearch	R	x		State of SkipBeat Second search function. When set, indicates that Second search is completed.
1	SbFinalSearch	R	x		State of SkipBeat Final search function. When set, indicates that Final search is completed.
0	SbAdjust	R	x		State of SkipBeat Adjust function. When set, indicates that Adjustment is completed.

2.20.14 Qsc Lane Control Register

Register

R_QscLaneControl

Attributes

-kernel

Address

0xE_6D00_0098

Bit	Mnemonic	Access	Reset	Type	Definition
7	ClrLaneHealth	RW	0		Clear lane health. For every transition of 0-to-1 of this bit, lane health bit of lane is cleared.
6					Reserved.
5	ForceSkipBeat	RW	0		Force Skipbeat. For every transition of 0-to-1 of this bit, RxClk offset is skipped 1-bit time. This field is intended to be used in manual setting of RxClk. This bit should be clear after manual setting of RxClk is completed.
4	SkipBeatEnable	RW	0		Skip Beat Enable. At the transition from 0-to-1, SkipBeat function is executed once using value selected in “SkipBeatOffset”.
3:0	SkipBeatOffset	RW	0x5		SkipBeat Offset. The receiver RxClk offset is equal to “SkipBeatOffset” bit-time wrt sclk. The power-on default value is 5(hex). This field is 4-bit wide and SkipBeatOffset can be selected from 0(hex) to 9(hex). The values in this field are modulo-10. For applying newer value of SkipBeatOffset, SkipBeatEnable should be toggled.

2.20.15 R QscRxBBDiag

Register

R_QscRxBBDiag

Address

0xE_6D00_00a0

Bit	Mnemonic	Access	Reset	Type	Definition
4	BBEnab	RW	0		Bit Blasting mode enable.
3:1	RxPattern	RW	0		Receiver bit-blasting pattern type. (a) 0x0 - repeat k28.5 (PNULL) 31 times and k28.0 (ANULL) (once) (b) 0x1 - PNULL (k28.5) (c) 0x2 - D10.2 (0x4A) (d) 0x3 - D24.3 (0x78) (e) 0x4 - IKJPAT pattern to stimulate inter-symbol interference (ISI) in ac-coupled system. Loop of 484 Character: ..... D30.3 (0x7E) 167 times ..... D20.3 (0x74) once ..... D30.3 (0x7E) once ..... D11.5 (0xAB) once ..... D21.5 (0xB5) 51 times ..... D30.2 (0x5E) once ..... D10.2 (0x4A) once ..... D30.3 (0x7E) 4times ..... D30.7 (0xFE) once .....D20.7, D11.7 (0xF4EB) 128 times
0	RxLaneSel	RW	0		Receiver Lane Select for bit-blasting patterns. When set, selected lane is enabled to check bit-blasting pattern type selected by RxPattern field. When clear, selected lane does not check for RxbbPattern.

2.20.16 R QscRxBBDiagStatus

Register

R_QscRxBBDiagStatus

Attributes

-noregtestcpu

Address

0xE_6D00_00a4

Bit	Mnemonic	Access	Reset	Type	Definition
1	RxLaneSync	R	x		Lane synchronization status. This bit will be set in BBMode, if RxLaneSync e is selected to check for RxbbPattern and finds RxbbPattern. This bit will remain clear if BBMode is not selected.
0	RxBBError	R	x		Bit Blasting error. This bit will be set in BBMode, if RxLaneSync is set and then if selected lane detects BBPattern error. Otherwise this bit will remain clear. This bit will also remain clear if RxLaneSelect bit is clear.

2.21 Link Unit Implementation Interface

Following sub-sections list handshake signals to and from link unit.

2.21.1 Interrupt Interface

The "fl_csw_Int_sa" is interrupt generating output signal from link unit. All Link interrupts are communicated by asserting this output. Refer to CSR section-2.20.11 for further details on interrupts from link unit.

2.21.2 Serial Configuration Bus Interface

The fabric link registers are acessible through the SCB interface. To connect to the SCB, a module must instantiate an SCB slave module, and connect it to a global SCB chain. The input is connected to chaini_scbs_dat_sr and the output is connected to scbs_chaino_dat_sr. The SCB bus and the SCB slave module are documented in the serial configuration bus chapter.

2.21.3 Differential Drivers and Receivers

A link unit drives 27 serial signals on 27 differential drivers and it receives 27 serial signals on 27 differential receivers. The differential drivers and receivers are part of Analogbit's QPMA and are described in Analogbit's document "Serdes PMA Programmer's Reference Manual".

2.21.4 Fabric Switch Interface

Following 8B10B characters will be used on serial lanes.

1. k28.0 (byte = 8'h1c) - alternate NULL character
2. k28.1 (byte = 8'h3c) - SOLS, start of LinkSync used by LinkSync hardware execution routine
3. k28.2 (byte = 8'h5c) - EOLS, end of LinkSync used by LinkSync during by hardware execution routine
4. k28.3 (byte = 8'h7c) - SOP, start of packet, used during MissionMode operation
5. k28.4 (byte = 8'h9c) - EOP, end of packet, used during MissionMode operation

6. k28.5 (byte = 8'hbc) - NULL or IDLE character
7. Following 8B10B control characters are Reserved. These 6 characters will be verified as part of data pattern verification cycle in LinkSync hardware execution routine but they are not used in Sicortex serial lane protocol. k28.6, k28.7, k23.7, k27.7, k29.7, k30.7

The Figure-2.10 shows handshake signals between fabric switch and link interface at the transmitter and at the receiver. The figure assumes that ICE9-A is the link transmitter of data packets and link receiver of control packets and ICE9-B is the link receiver of data packets and link transmitter of control packets.

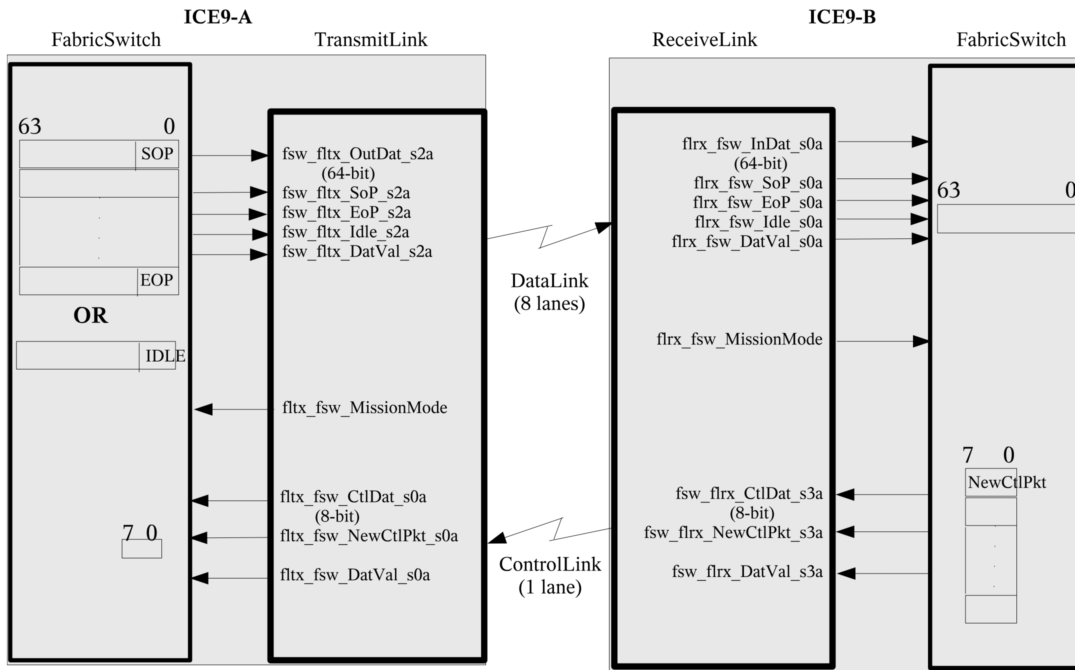


Figure 2.10: Handshake Signals

The data packets and IDLE packets, which are collectively referred to as data packets, begin at SwitchFabric of ICE9-A and travel from FabricSwitch to TransmitLink of ICE9-A on 64-bit wide data bus (fsw_fltx_OutDat_s2a). The TransmitLink transmits 64-bit data on 8-lane wide serial link which is connected to ReceiveLink of ICE9-B. The ReceiveLink of ICE9-B transfers 64-bit wide databus (flrx_fsw_InDat_s0a) and handshake signals to FabricSwitch of ICE9-B.

Correspondingly, the control packets begin at SwitchFabric of ICE9-B and travel from SwitchFabric to ReceiveLink of ICE9-B on a 8-bit wide databus (fsw_flrx_CtlDat_s3a). Then the link interface of ICE9-B transmits control and/or data characters on a single serial lane which is connected to ICE9-A. The link interface of ICE9-A will transfer 8-bit databus (fltx_fsw_CtlDat_s0a) and handshake signals to FabricSwitch of ICE9-A.

2.21.5 The transmitter Handshake Ports

Signal Name	From	To	Description

fltx_fsw_MissionMode	TransmitLink	FabricSwitch	When clear: (a) TransmitLink is down and not available for transmitting data or control packets. (b) FabricSwitch must not assert fsw_fltx_DatVal_s2a signal. If fsw_fltx_DatVal_s2a signal is asserted and if fltx_fsw_MissionMode is clear, then transmitter link may drive unpredictable characters on serial link causing unpredictable behavior at the receiver. When set: (a) TransmitLink is up, available for transmitting data. (b) If fsw_fltx_DatVal_s2a signal is clear then TransmitLink will drive either NULL (k28.5) or alternate NULL (k28.0) characters on all 8 lanes. An alternate NULL (k28.0) character will be driven every 8th cycle of fsw_fltx_DatVal_s2a signal remaining deasserted. (c) If fsw_fltx_DatVal_s2a is set then TransmitLink will transmit either control or data characters on serial lanes. The status encodings of fsw_fltx_SoP_s2a, fsw_fltx_EoP_s2a, and fsw_fltx_Idle_s2a are valid if among those 3 signals, condition of mutual exclusion is met and the rest of the bytes in FORD are data bytes
fsw_fltx_DatVal_s2a	FabricSwitch	TransmitLink	Data Valid signal. When set: Indicates that control signals fsw_fltx_SoP_s2a, fsw_fltx_EoP_s2a, fsw_fltx_Idle_s2a, and data bus fsw_fltx_OutDat_s2a[63:0] are valid. When clear: Control signals and data bus are invalid.
fsw_fltx_OutDat_s2a	FabricSwitch	TransmitLink	64-bit data bus, FORD = {Byte7,Byte6,..Byte0}
fsw_fltx_SoP_s2a	FarbricSwitch	TransmitLink	Start of packet, ignores BYTE0 and sends control character k28.3 on lane0. Ignore start of packet if either fsw_fltx_EoP_s2a or fsw_fltx_Idle_s2a is also set.
fsw_fltx_EoP_s2a	FabricSwitch	TransmitLink	End of packet, ignores BYTE0 and sends control character k28.4 on lane0. Ignore end of packet if either fsw_fltx_SoP_s2a or fsw_fltx_Idle_s2a is also set.
fsw_fltx_Idle_s2a	FabricSwitch	TransmitLink	Idle packet, ignores BYTE0 and sends control character k28.5 or k28.0 on lane0 configured in CSR: 2.16.6. Ignore Idle packet if either fsw_fltx_SoP_s2a or fsw_fltx_EoP_s2a is also set.
fltx_fsw_CtlDat_s0a	TransmitLink	FabricSwitch	8-bit databus
fltx_fsw_NewCtlPkt_s0a	TransmitLink	FabricSwitch	When set, indicates marker for new control packet and databus fltx_fsw_CtlDat_s0a = 8'h7c
fltx_fsw_DatVal_s0a	TransmitLink	FabricSwitch	When fltx_fsw_MissionMode is clear then, (a) Fltx_fsw_DatVal_s0a will remian deasserted (b) fltx_fsw_CtlDat_s0a and fltx_fsw_NewCtlPkt_s0a are invalid and should be ignored. If fltx-fsw_MissionMode is set then, (a) If fltx_fsw_DatVal_s0a signal is clear then it indicates that on serial lane neither valid data nor SOP was detected and hence fltx_fsw_CtlDat_s0a and fltx_fsw_NewCtlPkt_s0a must be ignored. (b) If fltx_fsw_DatVal_s0a is set and if fltx_fsw_NewCtlPkt_s0a is set then it indicates marker for new control packet (fltx_fsw_CtlDat_s0a = 8'h7c) otherwise fltx_fsw_CtlDat_s0a has valid data byte.

2.21.6 The Receiver Handshake Ports

Signal Name	From	To	Description
flrx_fsw_MissionMode	ReceiveLink	FabricSwitch	When clear, (a) ReceiveLink is down and not available for receiving data and transmitting flow control packets. (b) flrx_fsw_DatVal_s0a signal will remain de-asserted. Rest of the handshake signals, flrx_fsw_SoP_s0a, flrx_fsw_EoP_s0a, flrx_fsw_Idle_s0a, and flrx_fsw_InDat_s0a are invalid and must be ignored. When set, (a) If flrx_fsw_DatVal_s0a is clear then the rest of the handshake signals are undetermined (may be ignored). (b) If flrx_fsw_DatVal_s0a is set then rest of the handshake signals are valid. (c) Flow control signals fsw_flrx_* are valid signals.
flrx_fsw_InDat_s0a	ReceiveLink	FabricSwitch	64-bit databus FORD = {Byte7,Byte6,..Byte0}
flrx_fsw_SoP_s0a	ReceiveLink	FabricSwitch	Start of packet, (Byte0 = 8'h7c)
flrx_fsw_EoP_s0a	ReceiveLink	FabricSwitch	End of packet, (Byte0 = 8'h9c)
flrx_fsw_Idle_s0a	ReceiveLink	FabricSwitch	Idle packet, (Byte0 = 8'hbc)
flrx_fsw_DatVal_s0a	ReceiveLink	FabricSwitch	If flrx_fsw_DatVal_s0a signal is clear, then rest of the handshake signals (flrx_fsw_SoP_s0a, flrx_fsw_EoP_s0a, flrx_fsw_Idle_s0a, flrx_fsw_InDat_s0a) are undetemined (and hence may be ignored). If flrx_fsw_DatVal_s0a is set, then rest of the handshake signals are valid. Following five conditions will drive flrx_fsw_DatVal_s0a signal to logic state 1. (1) (Byte7 through Byte1 have valid data) & Byte0 has control character k28.0 (2) (Byte7 through Byte1 have valid data) & Byte0 has control character k28.3 (3) (Byte7 through Byte1 have valid data) & Byte0 has control character k28.4 (4) (Byte7 through Byte1 have valid data) & Byte0 has control character k28.5 (5) (Byte7 through Byte0 have valid data)
fsw_flrx_CtlDat_s3a	FabricSwitch	TransmitLink	8-bit databus
fsw_flrx_NewCtlPkt_s3a	FabricSwitch	TransmitLink	When fsw_flrx_NewCtlPkt_s3a is set, it indicates marker for new control packet and the databus fsw_flrx_CtlDat_s3a is ignored. Serial lane will drive control character k28.3 on serial lane when this signal is set.
fsw_flrx_DatVal_s3a	FabricSwitch	TransmitLink	When flrx_fsw_MissionMode is clear then, (a) ReceiveLink is down and not receiving FORDs or transmitting control packets. (b) FabricSwitch must not assert fsw_flrx_DatVal_s3a signal. If fsw_flrx_DatVal_s3a signal is asserted then link may drive unpredictable characters on link causing unpredictable behavior at the receiver When flrx_fsw_MissionMode is set then, (a) If fsw_flrx_DatVal_s3a signal is clear then TransmitLink will drive either NULL (k28.5) or alternate NULL (k28.0) character on a serial lane. (b) If fsw_flrx_DatVal_s3a signal is set then status encodings of the rest of the handshake signals are valid.

Chapter 3

The Dense Fabric Switch

[Last Modified $Id: fabric.lyx 43331 2007-08-15 18:23:44Z wsnyder $]

3.1 Overview

Each node chip contains a buffered crossbar switch which forms the basic element from which the SiCortex communication fabric is built. The switch is designed to provide the necessary components of a degree three Kautz network, though it would be well suited to building a 3-dimensional torus, fat tree, butterfly network, or other commonly used topology.

3.1.1 Specifications

- 3 input links, 3 output links. Input and output links do not, in general, connect to the same nodes.
- 2 GBytes/sec per link. 8 data lanes per link, plus a forwarded clock and a reverse channel which carries flow control information.
- Signaling: Max frequency 1 GHz, DC-balanced 8/10 code.
- Best case transit time through idle Fabric Switch: 15 ns. Transit time is measured from when start of packet is flopped-out by Link till when flopped-in by Link. Transit time increases for Dma-to-Link or Link-to-Dma packets, or if the downstream switch is congested.
- Maximum packet size: ~160 bytes (128 byte payload).
- Virtual Channels: 16.
- Buffers: 16 packets at each crosspoint.
- Ordering: Any packets with the same source node, destination node, and VC, following the same path, must remain in order.

3.2 Differences, Bugs, and Enhancements

3.2.1 Product and Chip Pass Differences

1. None.

3.2.2 Known Bugs and Possible Enhancements

1. The FSW has an architectural performance limit preventing 4 ford packets at max rate, bug1832.

3.3 Description

3.3.1 Routing

Packets are assigned fixed routes through the fabric by the originating node. The first FORD of each packet header contains a string of 2-bit routing codes. (See section 3.4.1.) At each hop, the receiving node examines the first routing code of the string, which selects among the three fabric link output ports or an escape. If one of the outputs is selected, the string is shifted right by 2 bits (one code) before transmission to the next node.

Routes are only shifted on packets traveling from IB to OB, not for packets going to or from the DMA.

3.3.2 Virtual Channel Assignment

Each packet is assigned a virtual channel (VC) number when it is created; the VC is chosen according to the path the packet is to follow, and determines the set of buffers available to the packet at each switch node along its path. The VC is encoded in the packet header. Each switch node has a programmable function which is able to conditionally decrement the VC on packets passing through the port. This function is enabled at fabric configuration time, and specifies for each port and each virtual channel whether or not to decrement VC.

Why do we do this? We use VCs to prevent deadlock in the network. Imagine a network of three nodes connected in a ring. Node A sends packets to node B, B to node C, and C to node A. Each node can forward a packet (pass it through) or consume it.

Assume that each node has space for exactly one packet in its input buffer. When a packet arrives in the input buffer it is examined and either passed along or consumed. A packet is never passed along from one node to the next unless the sending node knows there is space available in the receiving node's input buffer. If node A wants to send a packet to node B, then it holds the packet in A's input buffer until there is free space in B's input buffer. Then the data is sent, and A's input buffer is made free.

Now imagine that node A wants to send a packet to node C. It will send the packet first to B when B's input buffer is empty. B will forward the packet to C when C's input buffer is empty. Further complicate things by imagining that at the same time A wants to send packet to C, B wants to send to A, and C wants to send to B. In the first "cycle," B will recieve A's packet, C will receive B's packet, and A will receive C's packet. Notice what happens in the next cycle. A wants to forward the packet it just got from C and put it in B's input buffer. But B's input buffer is filled – it holds a packet destined for C which is stuck at B because C's input buffer is filled. Nobody moves. We're stuck in a deadlock.

Note that we could add more input buffers at each node, but that would just postpone the problem. If we had two input buffers, we could lock up the network by making sure we send two packets from each node to the node two hops away. All that it would take is for one node to delay emptying a destination buffer or some small network delay and all the buffers would fill. The problem is that there is a resource dependency that wraps around in a cycle. A can't be free until B is free, but B can't be free until C is free, and C can't be free until A is free, but A can't be free until...

This deadlock was a showstopper for many complicated topologies until Dally and Sites described a scheme called "virtual channels" in a 1988 paper. In this scheme they proposed adding "extra" buffers at each network node, but divided the buffers into classes. That is, buffer number 0 was devoted to virtual channel 0, buffer N to virtual channel N. Next they designated a specific virtual channel for every packet. A given packet would travel on this virtual channel from its source to its destination. We could apply this scheme to our three node system by saying that all messages starting at node C will be sent on virtual channel 1, while all messages from any other node will travel on channel 0. This breaks the circular dependency, since though A can't be free until B is free and B can't be free until C is free, C's destination (A's input buffer for channel 1) is never blocked since channel 1 carries all messages starting at C. (I wish I could do a movie of this one. Ask me to show you on the whiteboard if this doesn't make sense – mhr.)

So the first step in applying the virtual channel idea to our network is to identify all the cycles. It is the cycles that we want to break. But we have a network with 972 nodes. How many cycles are there? Probably a bazillion or so. Identifying them would be a bit of an issue, so here's what we do: Number all the nodes from 1 to 972. Each node K has three links coming from some other nodes (call them A,B, and C) and three links going to other nodes (call them R,S, and T). Each of these nodes has a number. A node is "less than K" if its node number is less than K's node number. Now consider a circular path through a bunch of nodes. Each of those nodes has a number. For at least one node P in the cycle P's upstream node in the cycle (the node that connects to P's input) and P's downstream node in the cycle (the node that connects to P's output) are BOTH less than P. (Draw a cycle of five or six nodes and number them. Note that you can't arrange the numbers such that there isn't some node P that

fits our description. If you could, then you could build a building such that climbing the staircase would eventually bring you back to the bottom of the staircase.)

So, now for every path through a node (from one of its three inputs to one of its three outputs) we can identify which paths fit our criteria of the upstream node and downstream node both being "less" than this node. There is at least one such path in every cycle within our network. Remember that the idea of virtual channels is to use buffer assignments to "break the cycle." The original VC concept assigned a VC to a packet at the start of its path and the VC was constant for the entire tour. We add a twist. We start a packet on some VC X. Each time it passes through a node on the route such that the upstream and downstream nodes are both less than the current node, we decrement the VC. That breaks the cycle. [1]

The last remaining trick is to make the initial VC assignment to each packet such that the VC doesn't get decremented so many times that it falls below 0. (We provide 16 virtual channels in the SiCortex fabric architecture.) The likely method we'll use is to count the number of times the VC will be decremented on a particular route, from start to finish. It may never be decremented. It can, at most, be decremented no more than (L-1)/2 times for a route L hops long. So, for a network with a diameter of 7, we need no more than 3 virtual channels. We provide more than 3 so that some traffic can travel on channels 0,1,2,3 and other classes of traffic can travel on 4,5,6,7. I'm not sure why anymore.

The fabric switch can support 16 VCs, but if fewer VCs are needed, the extra buffers can be configured as a pool that is available to traffic on any VC. The PoolMask register specifies which buffers are dedicated and which are in the common pool. If 6 VCs are needed for a system configuration, set PoolMask to 0xFFC0 and only use VCs 0-5. The 16-bit value 0xFFC0 indicates that crosspoint buffer entries 0-5 are dedicated to VCs 0-5, and entries 6-15 are pool.

3.3.3 Virtual Channel Arbitration

So now we've got every packet assigned to some virtual channel. (And, we've noted, the VC may change as the packet flows through the network.) To avoid the network deadlock we need to provide a separate buffer on each node chip for each virtual channel. In fact, we go one better than this.

In our simple example of the three node ring, we had a set of VC buffers at each network input port. In the ICE9 chip, we have a set (16) buffers for each (input,output) pair. That is, traffic arriving at a node's port 0 and leaving on port 2 will go into a pool of buffers that is separate from traffic for any other pair of input and output ports. We call the crosspoint where traffic from input port X to output port Y a crosspoint buffer.

Each crosspoint buffer has a pool of 16 packet buffer entries. One crosspoint buffer (XB) is associated with each input port and output port pair, and the XB keeps track of:

1. the order in which the packet in each buffer arrived, and
2. the virtual channel to which that packet is assigned.

When a packet is in the XB it waits until the XB knows that there is space for it in the downstream node. For example, let's say that a packet arrives on port 0 of node K and is destined to leave on port 2. (We know this from looking at the routing instruction.) We also know from the routing instruction that when the packet gets to the downstream node D it will leave on – for example – port 1. Then the XB02 (the crosspoint buffer receiving data from input port 0 and sending it out on port 2) on node K looks at the "buffer busy mask" for XB?1 on the downstream node.[2]In our example, let's say that the packet is traveling on VC 3. XB02 asks "Is slot #3 in XB?1 on node D empty?" If so, then XB02 can send the packet on to node D and be assured that there is a place for the packet to go. In fact, since XB02 knows that the packet will go into slot 3, it sets the XBE_ENTRY field in the outgoing packet to tell node D to store the packet in slot #3.

As I noted above, there are 16 slots in the XB packet store. We only use 6 to 8 virtual channels. Slots 0 through N in the XB are dedicated to VCs 0 to N. (N is defined in the POOLMASK that is set via the FSW POOLMASK configuration register that is reachable from the CSR interface. See section 3.9.)

At all times, the XB has a conservative estimate of the available buffers in each of four XBs of the downstream switch (strictly, the available buffers in the XBs for the input port on the downstream node to which this XB's output link is connected). If there are free buffers in the pool (buffers not assigned to a specific virtual channel), the output port selects the oldest packet among its buffers (if the age is known only among packets from the same

[1] We're in the process of patenting the scheme I've described here, so please, no matter how dull the conversation might get at your next party, keep this whole story within the SiCortex community.

[2] I say XB?1 with the ? because we don't know the input port number that the message arrived on at node D. As it turns out, we don't care. K's XB02 can only cares about the four XB's in D that are connected to port 2 on node K.

input port, the output port should select among occupied input port buffers on a round-robin basis). If there are no free buffers in the pool, then only those packets for virtual channels known to have available buffers should be allowed to arbitrate, and the oldest such packet should be chosen. Once a packet is sent from an output port, the output block tells all XBs connected to it that the assigned buffer is busy.

The local picture of which downstream buffers are busy is maintained in the output block. Buffers get added to this list when they are sent downstream. Buffers get freed from this list when a control packet arrives from the downstream node with a new "link sequence number." We don't use packet-by-packet ACKs to signify correct reception, as this would be rather inefficient. Instead, the downstream link continuously sends control packets. One field in each control packet carries the sequence number of the last correctly received packet. The upstream node then frees up any entries in its local list of busy buffers that were consumed by the acknowledged packets.

But the local picture is not complete. The downstream node includes a set of "busy masks" in the same packet with the last good link sequence number. There are four such masks, one for each of the *downstream* XB's connected to this link. So, the output block maintains four local busy masks and receives four downstream busy masks. For each downstream XB, the OR of the local and the downstream mask yields a conservative picture of which buffer entries are free on the downstream node. The local busy mask contains a 1 for each packet in the replay buffer that hasn't been acknowledged. As soon as the packet is acknowledged, the local busy bit is cleared.

3.3.4 Flow Control

At the link level, each transmitter assigns a link sequence number (LSN) to every outgoing packet, and includes that number in the header. The receiver includes the most recently received sequence number (of an error-free packet) in its buffer status reports flowing up the reverse channel. The transmitter (which receives the buffer status reports) retains transmitted packets in a replay buffer, deleting a packet from the replay buffer when the downstream node indicates that it has been successfully received.

Of course, since we're using the LSN as an acknowledgement mechanism, we have a bit of a startup problem. Imagine that the transmitter sends its first packet with an LSN of 0. Now imagine that the first packet is corrupt. Here's the problem: the downstream node probably sent a control packet to the upstream node even before the first packet (LSN = 0) arrived. That control packet had to have *something* in the "last good LSN received" field. If it was 0, then we're already fouled up and we haven't even sent a whole packet yet. So, we start the transmitter at LSN=2 and start the receiver's last good LSN register at zero.

3.3.5 Error Control

First, a point to remember: a single-cabinet system has 972 nodes, each with three links consisting of 10 signal pairs, for a system total of 29160 signal pairs. Operating at 2x10ˆ9 bits per second, we have approximately 6x10ˆ13 bits transmitted and received per second, so for any practical bit error rate, our system will encounter signaling errors hundreds of times per second. It is therefore essential that we recover quickly and gracefully from the vast majority of them.

Packet error detection and recovery is performed at the link level. Each receiver calculates a packet checksum, and verifies it against the checksum provided by the transmitter. The receiver's status reports back to the transmitter include the sequence number of the most recently received packet, if there has been no error, or the last packet received prior to an error if there has been an error.

In the event of a detected error, the receiver notifies the transmitter of the error, and the transmitter re-sends all the packets following the last one correctly received. The output port contains a replay buffer (16 packets in size), and in the event of an error, rather than re-arbitrating for the packets in the crosspoint buffers, packets replay from the appropriate point in the replay buffer. This means that crosspoint buffers can be released at the time they win arbitration for the output port, and do not have to wait for correct receipt. It results in an estimated error recovery time of about 70 ns.

When operating smoothly, the fabric will achieve cut-through, meaning that the header of a packet will leave a node's output port before any error is detected, so a faulty packet may propagate through the network to its destination. To deal with this problem, the type code in the tail of the packet includes a "poison" code which is set at the node which first detects the error, and causes the packet to be discarded when it arrives at a destination. Packets that develop uncorrectable ECC errors while stored in a packet buffer will also be tagged as "poison."

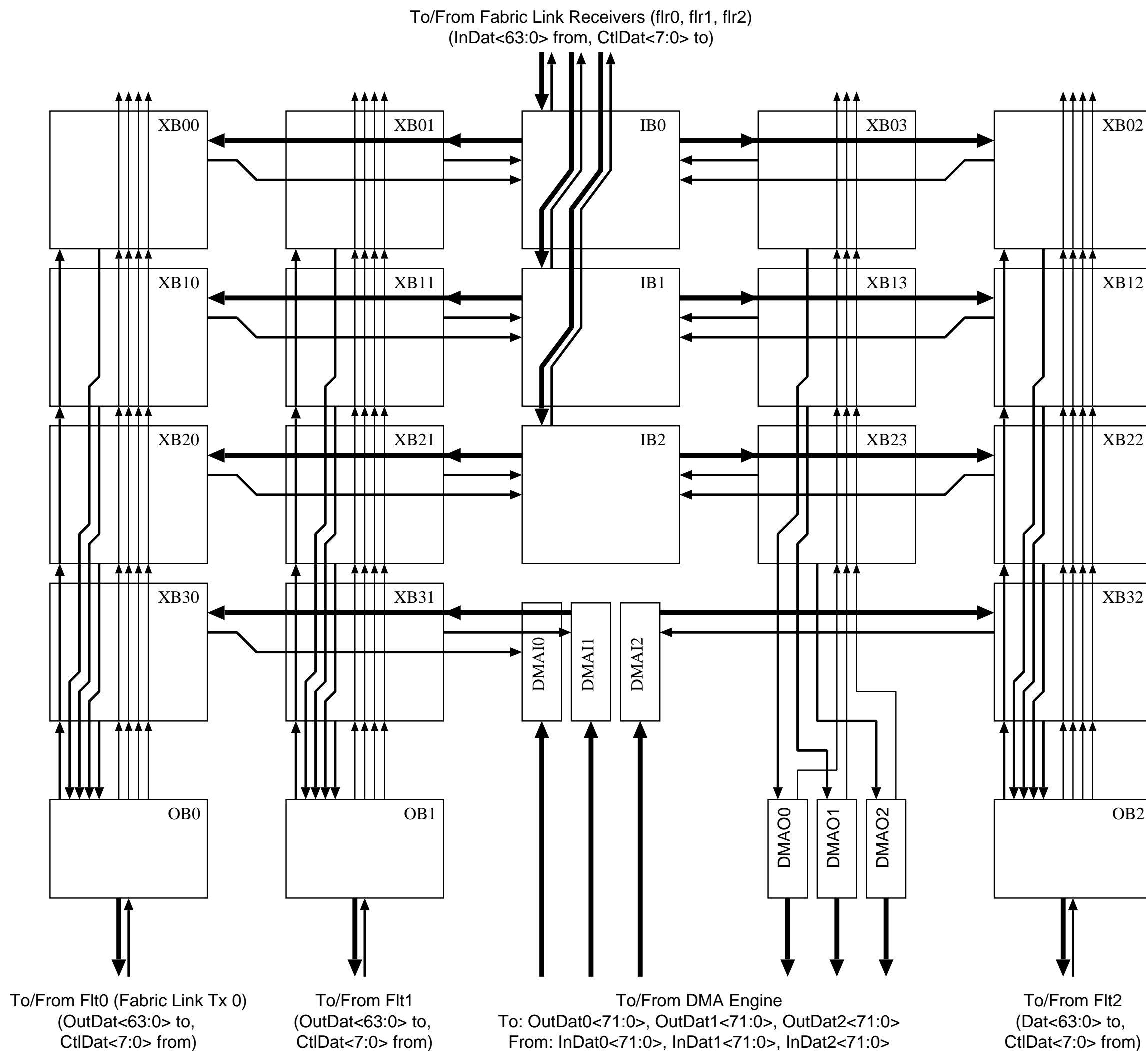


Figure 3.1: Fabric Switch Block Diagram

3.3.6 Out-of-Band Channel

Both upstream and downstream channels carry a specialized out-of-band fields in the packet which deliver one byte plus handshake information to the immediate neighbor. The byte is deposited in a software-accessible register in the neighboring switch's control registers, and the handshake bits maintain the full/empty status of the corresponding register in the source node. Whenever the handshake bits from the upstream or downstream nodes change value, an interrupt may be requested. There will be six such registers in each node, one for each upstream and downstream neighbor. This mechanism allows bidirectional out-of-band communication between neighboring nodes. It will be used at least for software configuration and management of the fabric, and we can implement TCP/IP on it if needed. (See sections 3.5.1, 3.4.1, and 3.4.2.)

3.4 Operation

When a packet arrives at the input port of the node (see **??** on page ??) it is re-timed into a sequence of 64 bit FORDs delivered at 1/5 the clock rate of the fabric (and is resynchronized/forwarded into the node's own Switch Clock domain.) All recoding and re-timing is handled by the node link receiver. The switch will then route the incoming packet to an appropriate output port via one of 15 crosspoint buffers.

The first FORD in each incoming packet (arriving at an input block or IB) contains the virtual channel number for the packet and the number of the port from which it will leave the switch. Ports 0, 1, and 2 lead to the three neighbor nodes, while port 3 leads to this node's DMA engine. The input block may decrement the VC number in

the header FORD (see 3.6) before sending the packet on to one of four crosspoint buffers. (This is based on the deadlock avoidance mechanism described below.) The link recognizes the first FORD in a packet by the presence of a start-of-packet marker in lane 0 of the link.

When a packet arrives at a crosspoint buffer, it is written into a free crosspoint buffer entry (XBE) at the location specified in the first FORD of the packet. (If the location is already occupied, the packet is dropped and marked "invalid.") Arriving packets are immediately bypassed to the output block if the crosspoint buffer is not otherwise sending data to the OB. If the output block is not currently moving a packet, it will pass the bypassed packet directly to the link output port. Otherwise, the packet will sit in the XBE until it wins a bid for transmission and is read from the crosspoint buffer. The last FORD in each packet contains an end-of-packet marker in its 8/10 encoded form on lane 0.

The output block is responsible for global arbitration among the packets offered by each of the four attached XBEs. We want to make sure that packets can be routed back to back, so that if XB00 is sending a packet through the output block (OB0), then we can immediately follow the end of its packet with the start of a packet from any of the XBEs. We do this by starting global arbitration a few cycles before the end of a packet is transmitted, so that a winner is ready in time to fill the next output cycle.

A few elements are not shown in the diagram because they touch nearly everything. The fabric switch contains a module FswCsr which connects to the Serial Control Bus (SCB). Control register values are distributed from FswCsr to every module, and status values are sent from every module back to FswCsr. The FswCsr can assert interrupts to notify processors of an error conditions or when an out-of-band character is received.

3.4.1 The Data Link

The data link is implemented with eight lanes of SERDES, differential, low-swing, channels, each passing bits between Ice-9 chips at 10-times Ice-9's internal clock rate. Inside Ice-9, on the interface between each Link unit and the Fabric Switch unit, one 64-bit Ford is passed on each clock. Each external lane handles 8 bits out of those 64 bits. A series of these Fords represents the flow of Data Packets or Idle Packets.

On any given interface between the Fabric Switch unit and a Link unit, when the signal indicating "valid data" is asserted, Data Packets or Idle Packets are passing. The Fords themselves don't indicate boundaries between packets, separate control signals say what each Ford is. The "SOP" signal indicates the Header, the first Ford of a Data Packet. The "EOP" signal indicates Trailer, the last Ford of a Data Packet. The "Idle" signal indicates this Ford is an Idle Packet (Idle Packets are 1 Ford long). Only one of these three signals may be asserted at a time, and if none are asserted, this Ford is in the middle of a Data Packet.

The format of a Header Ford, a Tailer Ford, and how they join with payload Fords to form a Data Packet are shown below. Also shown is the format of an Idle Packet.

The encodings of Sop, Eop, and EsComma fields are only meaningful in the 10-bit form, while on a differential link between Ice-9 chips. Inside Ice-9, between Links and Fabric Switch, we rely on separate SOP, EOP, and Idle control signals. A TX Link will ignore whatever value the Fabric Switch put in Ford bits 7:0 during the assertion of SOP, EOP, or Idle, and just manufacture the appropriate 10-bit control character to send over the differential link. An RX Link will put 8-bit values into Ford bits 7:0 when it receives Sop, Eop, or EsComma 10-bit characters, but these 8 bits are ambiguous, being the same as certain other ordinary data bytes. Fabric Switch knows these are not ordinary data bytes because of the control signals.

Although Link units are free to represent EsComma with either NULL or ANULL characters, which according to the 8b/10b encoding we use would produce different 8-bit values when decoded, the RX Link always puts the 8-bit value for NULL into bits 7:0 when decoding either NULL or ANULL.

Crc32 is created in a manner that doesn't include Sop, Eop, and EsComma fields, and when judging incoming Crc32's, those fields are again excluded.

3.4.1.1 Fabric Packet Header Class

Class

FswPktHdr

Attributes

Bit	Mnemonic	Type	Constant	Definition
w0[7:0]	Sop			Start of Packet.
w0[11:8]	Vc			Virtual Channel <3:0>.
w0[15:12]	XbeTarget			XBE Target <3:0>.
w0[21:16]				Reserved
w0[26:22]	NumFords			How many fords in the packet? Valid values are 4 to 20.
w0[27]	HasCtrl			This bit is only used by the DMA engine. If 1, the DMA treats the second ford as a DMA control ford, otherwise it is treated as payload.
w0[31:28]	Lsn			Link Sequence Number <3:0>.
w0[63:32]	Route			Route <31:0>.
w0[63:0]	AllBits			Header doubleword. Overlaps allowed.

3.4.1.2 Fabric Packet Trailer Class

Class

FswPktTrail

Attributes

Bit	Mnemonic	Type	Constant	Definition
w0[7:0]	Eop			End of Packet.
w0[11:8]	Type			Packet type (1111 = FSW_POISON_TYPE).
w0[15:12]	ProcessIndex			Process Index.
w0[31:16]	UnixProcessId			UNIX Process ID.
w0[63:32]	Crc32			CRC-32.
w0[63:0]	AllBits			Trailer doubleword. Overlaps allowed.

3.4.1.3 Fabric Data Packets

Ford	Bits	Content
0	63:0	Header (as shown above)
1 to (last-1)	63:0	Payload, including software header
last	63:0	Trailer (as shown above)

3.4.1.4 Fabric Packet Idle Class

Whenever an output port has no data packets to transmit, it sends an Idle packet, consisting of a single ford encoded as shown.

Class

FswPktIdle

Attributes

Bit	Mnemonic	Type	Constant	Definition
w0[7:0]	EsComma			ES_COMMA (K28.5 NULL or K28.0 ANULL).
w0[15:8]	OutOfBand			Out of Band Byte.
w0[16]	EmptyFlag			Empty flag.
w0[17]	TakenFlag			Taken flag.
w0[18]	ErrorAck			Error Acknowledge.
w0[31:19]				Reserved.
w0[63:32]	Crc32			CRC-32.
w0[63:0]	AllBits			All bits of Idle Packet. Overlaps allowed.

3.4.2 The Control Link

The control link has one lane, one differential pair between Ice-9 chips, 8-bits wide between the Fabric Switch unit and Link units. Control Packets are 15 bytes long.

Between Ice-9 chips the start of a Control Packet is indicated by the 10-bit SOP character. Between Fabric Switch and Link units, the start of a Control Packet is indicated by a NewCtlPkt control signal. This is because the 8-bit encoding of the 10-bit SOP character is ambiguous, being the same as another ordinary data byte. A Link unit sending a Control Packet will ignore b0. A Link unit receiving a Control Packet will put the 8-bit encoding of SOP into b0, and assert NewCtlPkt at that time. Note that CSUM intentionally does not cover the SOP field.

3.4.2.1 Fabric Control Packet Class

Class

FswCtlPkt

Attributes

Bit	Mnemonic	Type	Constant	Definition
b0[7:0]	Sop			Start of Packet. During CRC computation, assume SoP=0.
b1[3:0]	Lsn			LSN
b1[4]				Reserved
b1[5]	ErrFlag			Err flag
b1[6]	TakenFlag			Taken flag
b1[7]	EmptyFlag			Empty flag
b2[7:0]	P0BusyHi			P0Busy[15:8]
b3[7:0]	P0BusyLo			P0Busy[7:0]
b4[7:0]	P1BusyHi			P1Busy[15:8]
b5[7:0]	P1BusyLo			P1Busy[7:0]
b6[7:0]	P2BusyHi			P2Busy[15:8]
b7[7:0]	P2BusyLo			P2Busy[7:0]
b8[7:0]	P3BusyHi			P3Busy[15:8]
b9[7:0]	P3BusyLo			P3Busy[7:0]
b10[7:0]	Oob			OOB
b11[7:0]	Crc3			Running CRC of bytes 0-10. During CRC computation, assume Crc3=0.
b12[7:0]	Crc2			Running CRC of bytes 0-11. During CRC computation, assume Crc2=0.
b13[7:0]	Crc1			Running CRC of bytes 0-12. During CRC computation, assume Crc1=0.
b14[7:0]	Crc0			Running CRC of bytes 0-13. During CRC computation, assume Crc0=0.

3.4.3 Control Link Use

It is a good idea, in the case of a critical flow control scheme, to assume that packets will be dropped, corrupted, spindled, mutilated, or otherwise ill treated on their way from source to sink.

As we discussed earlier, flow control is managed by a debit/credit mechanism where the receiving end tells the sender how much space is available in the receiver's buffers for each virtual channel and port. As you might imagine, this scheme works fine in the presence of imperfect knowledge on the part of the transmitter, as long as the transmitter's view of the world is always pessimistic: the transmitter must never send a packet for which there is no room at the receiver.

The receiver will provide this imperfect information by keeping up a continual chatter on the control link. Each control packet begins with an `ES_COMMA` tenbit character. Section 3.4.2.1 describes the layout of the control packet. Each control packet will contain the serial number of the last packet received without error. We'll call this the "link sequence number" or *LSN*. As each packet arrives intact on the data link, the receiver updates the LSN and

it is sent back in the next control packet. If an error is detected in an arriving packet, the receiver will not update the LSN and will set the Err_Flag entry in the control packet. (The LSN field holds the link sequence number from the last successfully received packet.) The error flag remains set in all subsequent control packets until a data link message arrives indicating an error recovery retransmission. (This is done via the Error Acknowledge bit in Idle packet). The other bits sent along with the LSN are used to manage the out-of-band communication channel described below.

In addition to the LSN, the control chatter needs to update the availability of up to 16 virtual channels and a shared buffer pool for each of the four outlets at the end of a data link. (See 3.7 for a discussion of buffer allocation in the fabric switch.) Each outlet for a switch has 16 buffer slots. The interpretation of a buffer slot name vs. the virtual channel to which it belongs is programmable – the meaning is determined by agreement between the downstream and upstream nodes on a link. The control link protocol only specifies the means of identifying the occupancy for each of the 16 entries in each of the four crosspoint buffers. The control packet stream carries a current snapshot of the crosspoint buffer entry utilization for each of the four crosspoint buffers. Each XB (crosspoint buffer – see 3.10.7) has 16 entries. The arbitration unit within the switch determines which packets may be forwarded to the next node on a path based on the availability of downstream buffer entries. If bit **N** is set in **PxBusy**, buffer slot number **N** on crosspoint buffer **x** is currently filled.

Finally, we provide a "out-of-band" communication link between nodes that travels along the control link. The out of band link is described in section 3.5.1.

3.4.4 Error Recovery

Note that we're not doing error correction. It turns out that error correction on a 10/8 code is rather expensive. Parity based schemes (including most SECDED codes) rely on the likelihood of a single bit error being much greater than a multi-bit error. Unfortunately, it is unlikely that we could construct a mapping from the tenbit space into the eightbit space that preserves the error bit count. That is, for some ten-bit combination, there will be a single bit error in the encoded symbol that will result in two or more bits in error for the decoded symbol. Consider a symbol with an equal number of 1's and 0's. Each of the ten possible single bit errors will create a new symbol with either six 1's and four 0's or vice-versa. Each of these ten distance one symbols will decode to some legal eightbit value. At least one of those must differ in at least two bits from the original (correct) decoded value, because there are only eight values that are distance 1 from the original value. So the first alternative is a symbol correcting code that could correct one bad symbol out of 255. The cost of the symbol correction hardware really isn't worth it.

Simple linear codes are hopeless here. So we'll use a CRC error detection code.

Every data packet is protected by a CRC error-detection code, and every output port has a replay buffer in which it records the data packets recently sent on that link (Idle packets are not recorded). The connected input port records the Link Sequence Number (LSN) of every packet, and as long as the CRC's are correct, sends the LSN's back to the output port via periodic control packets. In the event of a CRC error, the input port stops updating the returned LSN and instead reports an error in the control packet. The output link uses the LSN of the last correctly-received packet to look up the position of the erroneous packet in the replay buffer, and re-sends the corrupted packet and all its successors. When the downstream node receives the retransmitted packet, if the CRC is correct it updates the returned LSN, and the output link resumes taking packets from the switch when it has finished retransmission from the replay buffer.

Control packets and idle packets are never replayed. The switch that generates these packets creates new packets with a new CRC constantly. The switch that receives these packets simply ignores packets which have a bad CRC and waits to receive the next one.

3.4.5 Poison

When the header of a packet arrives at an input port, the switch immediately arbitrates for use of the selected output port, and if it's available, begins outputting the packet. This is called cut-through routing, and is an important contributor to the performance of the SiCortex fabric. However, it creates a problem if the packet contains errors that aren't apparent from the first ford; for example, a packet length error cannot be detected until we realize that the packet is too long. There is no way to prevent the header from continuing on to its destination – or if corrupted, some other destination altogether. The solution to this problem is to "poison" the packet. Any node which detects an error will change the Packet Type field in the packet's last ford to Poison (and increment a counter of how often it has poisoned packets). When the packet finally arrives at some destination, the Poison type will be recognized and counted, but the packet will be otherwise ignored. Switch nodes which buffer a poisoned

packet waiting for an output port are permitted to discard the entire packet, provided that it has not begun output. For more detail on error recovery, see section 3.8.

3.4.6 Mission Mode

The fabric switch depends on the fabric link transmitter (FLT) and receiver (FLR) to send data to its neighboring nodes. While the FLT and FLR are being initialized and the link is in training, each link deasserts a signal to the FSW called MissionMode. While MissionMode is off, the fabric switch ignores everything else coming from that link, to avoid being confused by the training sequences. Once MissionMode is asserted by a FLR, the switch begins to accept data packets and send control packets. When MissionMode is asserted by an FLT, the switch waits for the first good control packet, then begins sending data packets downstream.[3]

3.5 Special Communication Paths

3.5.1 The Out-of-Band Communication Registers

It is quite handy to have a low bandwidth simple communications path between an upstream and downstream node. Normally the network topology would not allow communication from a downstream node **B** back to its upstream node **A** without requiring a message to pass through multiple hops.

Half of the sub-band communication path is implemented on the control link, the other half is in the data link. The control link carries OOB information in every control packet; the data link carries OOB information in Idle packets, when the link has no data packets to transport.

Each node has six OOB links: three to its upstream neighbors, and three to its downstream neighbors. Each OOB link is bidirectional, with a send and receive register at each end of the link. Let's assume that Empty is high and Taken is low. To use the link, software writes a byte to the send register, and clears the Empty flag. The fabric transmits the register and flag to the far end of the link as convenient (in Idle or Control packets), and when they are received without error, the far-end fabric switch writes both to the receive register. The receive register requests an interrupt when it sees the Empty flag toggle. Interrupt software on the far-end node reads the receive register, and sets the Taken flag, which then gets passed over the reverse channel and causes an interrupt on the source node. To return to initial conditions, software on the source node sets Empty again, which propagates and triggers an interrupt at the receiver. Then software on the receiver clears Taken, which propagates and triggers an interrupt at the source.

The Out-Of-Band communication is driven entirely by software, so other communication protocols may be possible as well.

3.6 Deadlock Avoidance

The fabric uses a virtual channel scheme to avoid network deadlock. For more information on virtual channels and deadlock avoidance, see Section 3.3.2. The fabric switch core makes no changes to virtual channel assignment for a packet. It is the responsibility of the input block to decrement the virtual channel assignment per the deadlock avoidance scheme.

For instance, consider a packet arriving at receiver block 0 on virtual channel 3 on a route that dictates a decrement of the VC number and will leave the chip on port 1. The upstream node has already verified that crosspoint buffer XB01 has room for a packet on VC3. The incoming packet will consume the appropriate slot in XB01 and arbitrate for access to a crosspoint buffer entry on the next chip for VC2. The choice of whether a packet's VC is decremented is made in the input block for a port. Each IB has a 3-bit DecrementVc register indicating which (if any) packets get a VC decrement *based on the output port* selected by the routing field at the head of the packet. If bit X of the DecrementVc register is set, then VC is decremented for packets whose destination is output port X. DecrementVc has only 3 bits because packets going to the DMA (output port 3) are never decremented.

[3]The link always delays the assertion of datValid two cycles after the assertion of mission mode. The DV infrastructure follows this implementation.

3.7 The Switch Architecture

3.7.1 General Organization

When an outbound link becomes available, each crosspoint buffer set (four crosspoint buffers connected to the same output port) must pick the best eligible crosspoint buffer entry to send out. The "best eligible" entry is, ideally, the oldest. Finding the oldest entry of 16 within a single crosspoint buffer is relatively straightforward and inexpensive. We call this stage "Local Arbitration." Once each of the four crosspoint buffers in a set (*e.g.* **XB00, XB01, XB02,** and **XB03** in Figure 3.1) has chosen a local candidate, the four local candidates bid against each other in the "Global Arbitration" phase.

3.7.2 Ordering Requirements

The global ordering rule dictates that packets from the same source, going to the destination along the same route, with the same virtual channel must be kept in order.

Another way to state the same ordering rule is: If there are any differences in the route or VC number, packets are allowed to pass each other. Our fabric switch does not bother to compare all the bits of route though; it only looks at the least significant four bits, which indicate the destination port in this fabric switch and the destination port in the downstream fabric switch. This is an implementation choice; there are other legal choices. Our implementation of the fabric switch keeps packets in order only if they

1. arrive on the same input port
2. leave on the same output port (route bits 1:0)
3. are destined for the same output port of the downstream switch one hop away (route bits 3:2)
4. leave on the same virtual channel

To maintain this ordering, every crosspoint buffer must keep a record of the relative age of all of its packets. We never need to compare packet age between crosspoint buffers, because they are on different routes.

3.7.3 Local Arbitration: Within A Crosspoint Buffer

When there is an opportunity for a packet to be sent out of an output port, each crosspoint buffer contending for that port selects its oldest eligible packet and sends a "bid" to the output port for that packet. Packets are eligible if there is a buffer in the downstream fabric switch which can accept the packet. The following paragraphs describe how the oldest eligible bidder is determined.

Each entry in the crosspoint buffer has a 16-bit wide "age vector" associated with it (where "16" is the number of entries in a crosspoint buffer). When a new packet arrives with XbeTarget = W, slot W is filled, and its age vector is set to all 1s except for bit W. At the same time, bit W in ALL the age vectors within this crosspoint buffer are cleared.

Only "eligible entries" are allowed to bid in a local arbitration cycle. An entry X is eligible if the busy mask bits from the output port indicate that there is a buffer entry Y at the destination link that can accommodate the packet in entry X. (That is, entry X – carrying a packet for port P and virtual channel V on the next node – is eligible only if the next node has space for a packet in XB?P.)

One cycle after each eligible entry has bid, each entry ANDs its age vector with the vector of bids. If the AND of the two is zero, then the corresponding entry wins the local arbitration. Only one such entry can occur for any given bid cycle.

A crosspoint buffer performs local arbitration in every cycle to select a local winner. If there is a local winner, the crosspoint buffer raises a request to its output block. The request consists of the following information about the local winner:

- Which type of request it is. There are two types of requests:
  - Request for Packet Store. If granted, the crosspoint buffer will start reading its memory and start sending the packet, one ford at a time, to the OB.

- Request for Bypass. If granted, the OB will read packet data from its bypass delay pipeline and send it out; the crosspoint buffer doesn't have to send anything. Bypass is only possible during a window of 3 cycles after the start of packet arrives, but during that window the bypass path provides a lower latency path through the switch.

- Virtual Channel. (Remember that any VC decrement has already been done in the IB.)

- Which output port will the packet use in the downstream switch?

- Which crosspoint buffer entry will the packet use in the downstream crosspoint buffer?

- How many fords in the packet?

The crosspoint chooses a local winner every cycle based on continually-changing information from several sources. The OB receives control packets and forwards the downstream buffer availability to the crosspoint buffer. When requests are granted, the winning packet is invalidated so that it doesn't arbitrate anymore. New packets arrive and begin to compete for a chance to be the local winner. Because the inputs are changing every cycle, the local winner may change every cycle and this is perfectly legal, but there's one caveat. When the grant signal comes from the output block, it always refers to the local winner that generated a request in the previous cycle.

The output block will ignore any requests that are made at incovenient times, such as during the grant cycle, during replay, or when the outbound link has gone down.

3.7.4 Global Arbitration: Between Crosspoint Buffers

In the previous section we saw that the crosspoint buffers will make requests to the output block in every cycle. Each OB sees at most four requests from the four connected crosspoint buffers. Whenever the output port is free, or is close to the end of a packet, global arbitration looks at the requests and decides what packet will be sent next.

Before describing global arbitration, we must consider what packets are really competing for. Before a packet can be sent downstream, the switch must be sure that an appropriate buffer is available for it in a downstream switch's crosspoint buffer. So, packets are competing for a spot in a particular crosspoint buffer; the packet's low 2 bits of route[4] tells which crosspoint buffer they need to go into when they get there. Also, within a downstream XB, some XBEs are dedicated to a VC while others can be used by any VC (see PoolMask register). In global arbitration, we try to ensure fair access to the downstream buffers. If requests from different XBs request the same NextPort (low 2 bits of route) and VC, they are contending for the same pool of buffers and must be treated fairly.

Global arbitration is done in two stages. The first stage selects the least recently chosen XB which is requesting. The first stage winner's NextPort and VC are used in the second stage. In the second stage, we only consider requests that have the same NextPort and VC as the first stage winner. Often, that narrows it down to just one, but there might be up to four requests remaining that all have the same NextPort and VC. The second stage does round-robin arbitration between remaining requests, based on just the history of requests with this same NextPort/VC combination. The winner of the second stage will be selected to go out the output port as soon as possible. The XB that wins is recorded in the stage 1 and stage 2 history so that it influences the next global arb.

The following diagram describes the state that is stored to implement the two arbitration stages in one output block.

[4]When the packet arrived in the IB, the low 2 bits of route told which output block to send it to. After looking at them, the IB shifted those 2 bits away. By the time the packet is in the crosspoint buffer, the low 2 bits of route tell which output block it will go to in the downstream switch.

Global Arbitration Example

Stage 1 Contenders

Three out of four XBs request.
The requests are shown, along with their VC and NextPort.

XB02, VC=4, NP=7
XB12, VC=0, NP=3
XB22 no request
XB32, VC=0, NP=3

Stage 1 Arbitration

Find Least Recently Chosen XB that is requesting. Use a 4x4 Age Vector Matrix.

XB02	0	1	1	0
XB12	0	0	0	0
XB22	0	1	0	0
XB32	1	1	1	0

This table shows the XB12 won least recently, followed by XB22, then XB02. XB32 won most recently. Eliminate the rows and columns for the one that is not requesting, and look for a row full of zeroes. XB12 is the winner.

Stage 1 Result

The stage 1 winner is:

XB12, VC=0, NP=3

Any requests with the same VC and NextPort will proceed to stage 2.

Stage 2 Contenders

There are two contenders which had the same VC and NextPort as the stage 1 winner.

XB12, VC=0, NP=3
XB32, VC=0, NP=3

Stage 2 Arbitration

Do Round Robin by NextPort+VC using a 64-entry Table of which XB won last.

vc 0, nextport 0	XB02
vc 0, nextport 1	XB32
vc 0, nextport 2	XB02
vc 0, nextport 3	XB12
vc 1, nextport 0	XB12
...	...
vc 14, nextport 3	XB22
vc 15, nextport 0	XB22
vc 15, nextport 1	XB12
vc 15, nextport 2	XB02
vc 15, nextport 3	XB32

Stage 2 Result

Last time there was a stage 2 arb for VC=0, NP=3, the table says that XB12 won. Give it lowest priority this time. XB32 is the winner.

XB32, VC=0, NP=3

Last step: Update the stage 1 age vector and stage 2 history table.

In the first stage, we need to know the least-recently-used crosspoint buffer that is requesting, so we maintain four 4-bit age vectors. The NextPort and VC of the first round winner are used to index into the stage 2 table, which records the previous winner for each combination of NextPort and VC. The second stage round-robin gives the previous winner the lowest priority in winning stage 2 this time. After a winner is chosen, the stage 1 age vector is updated, and the winner's XB number is stored in the appropriate entry of the stage 2 history table.

After a winner is chosen, the output port sends a Grant signal to the XB saying that the packet was selected to be transmitted. The XB knows that the grant applies to the request from the previous cycle. The XB clears the Valid bit on the entry that won, so that a new packet can begin to use that entry. If the request was a "Request for Packet Store", then the XB needs to start shipping data to the XB in the following cycle.

Back in the output block, the new winner has declared its XbeTarget along with the request, so the OB can set a bit in its local pessimistic view of downstream buffer availability. The local pessimistic view is logical ORed with the buffer busy mask sent to the crosspoint buffers, so that future requests will assume that the buffer is taken. Eventually, the OB receives an acknowledgment in a control packet, the pessimistic bit is cleared, and the downstream buffer can be used again.

When a winner is chosen, the OB knows how long the winning packet is and when it will be done. Therefore, it knows when to allow global arb to run again, just in time to select the next winning packet. Meanwhile, all requests are simply ignored by the OB. There is no reason that the XB needs to know if global arbitration is running or not. The OB only sends the XB a message if it wins.

When you combine local arbitration and global arbitration, it is easy to introduce the possibility of starvation. In several earlier implementations of output arbitration, we discovered cases where a certain traffic pattern in some XBs could prevent a packet in another XB from ever getting sent out. One important aspect of the scheme described above is that there is separate round-robin history maintained for the specific resources that packets are competing for. The NextPort and VC are used in stage 2 because each entry of the table exactly describes the set of buffers

that a packet needs. Using only the VC in stage 2 arbitration would allow a flood of traffic on (VC0, NextPort 0) to starve traffic on (VC0, NextPort 2). Using only the NextPort in stage 2 arbitration would allow a flood of traffic on (VC0, NextPort 2) to starve traffic on (VC1, NextPort2). Another important piece is that every requester must receive information on downstream buffer availability in the same cycle, so that if a buffer becomes available that allows an old packet to finally go out, it is guaranteed a chance to win local arb and eventually win global arb as well. Providing information at the same time is easy for the crosspoint buffers; we need to be especially careful in making bypass decisions. If the bypass decision logic learns of available buffers before the competing crosspoint buffers, bypass packets could starve normal traffic and break ordering and fairness rules.

3.7.5 Why Two Levels of Global Arbitration?

Matt wrote this section to describe some of the pitfalls of global arbitration schemes that didn't take into account NextPort and VC. Bryce left it in the spec because it describes one of the most important problems we're trying to avoid.

A single-stage "least recently chosen" scheme is fair, but not immune to livelock. Imagine that there is a packet X in XB00 that needs VC1 on port 3 in the next chip. At the same time all four XBs (XB00, XB01, XB02, XB03) are extremely busy and always have packets that are eligible to bid even when packet X can't. Now imagine that every time XB00 wins global arbitration, VC1 is busy, but XB00 has traffic for some other VC. This will ensure that every time VC1 becomes available to XB00, it is the least likely bidder to be chosen. (It wasted its turn on the traffic for the other virtual channel: the global winner is always XB01,XB02, or XB03 when there is space available in VC1/P3 of the next chip. The packet in XB00 for VC1 will never win the global bidding: it is stuck. Unlikely? Yes. Impossible? No. In fact, this bug has surfaced before.

The problem is that we're doing a two level arbitration where success for an individual requester requires success at both levels simultaneously. In the case of packet X, it won its own local bid whenever it was eligible (because it was eventually the oldest packet entry in XB00) but each time it got to bid on a global resource, other traffic in XB00 had caused the least-recently-chosen token to pass it by.

3.7.6 Stitching it all Together

It is important that we be able to string packets back-to-back through an output port. This means that as the last bits of a packet are being sent to an output port driver, we need to have the first bits of the next packet queued up and ready to go. To accomplish this, we have tuned the global arbitration logic so that it chooses a new winner several cycles before the data is needed at the output mux. By arbitrating several cycles before the end of packet is transmitted, we cover the delay of arbitration, notifying the winning XB, and starting to read the winning packet. This implementation does not require skid buffers.

3.8 Error Detection and Recovery

There are several places where bits could get flipped, slipped, spindled, or mutilated. As was indicated above, we attempt to isolate errors to the link level and retry in the presence of bit errors on the link. Some errors are recoverable, in the sense that we can retry the transmission and will get the bits across the link on the second or third try. Other errors may not be correctable in this way. In this latter case, we will "poison" the outgoing packet so that it will eventually be dropped into the bit-bucket somewhere along its future path.

It should be noted that some errors may be detected well after the packet has begun its trip to the next node on its path. That is, the head of a packet may have left a node before the tail has been seen and an error has been detected. This is a problem, and probably the only really tight path in the switch. The IB must detect an error in the last FORD (possibly a CRC error) and propagate a signal down to the OB in time to cause the OB to change the packet type in the last FORD of the packet to "POISON." This is serialized with the creation of the CRC field that is connected to the same packet. Note that we must generate good CRC for ALL transmitted packets, whether poisoned or not, otherwise we'll trigger a retry on the link, since all CRC mismatches are assumed to be caused by link transmission errors.

Nearly all error detection occurs in the input block, so that crosspoint buffers and output blocks do not have to worry about error conditions. The input block catches protocol errors such as missing or extra SoP or EoP markers, packets that are too long or too short to be legal, and CRC errors. Corrupted XbeTarget is detected in the crosspoint buffer so that avoid overwriting a good packet with a corrupted one. ECC errors are detected as a packet store or replay buffer entry are read.

3.8.1 CRC Generation and Checking

All packets (control, data, and idle) are covered by a 32 bit CRC. The algorithm is defined in Crc.sp. (This is the CRC-32 scheme.) The initial value for the CRC sum, before the first byte or word is cranked in, is 0xFFFFffff. The final value is NOT complemented before being written into the packet. All bits in the packet are covered by the CRC except for the SoP field from the link and the CRC value itself. (In the case of data packets, the top 32 bits of the last FORD are the CRC field, the low 32 bits are covered by the CRC.)

Many of the errors below are a subset of a CRC error. But some fields are more likely to confuse the switch than others. Bit errors in the payload are easy to handle. For the fields that the fabric switch really cares about, such as the VC, XbeTarget, and route, the recovery mechanism (if needed) is described in a separate section below.

3.8.2 Handling Poisoned Packets

Here is the problem:

Imagine a packet that is corrupted such that, while it had been traveling on VC 3, the VC got changed to VC 4. This is definitely a bad thing, since our deadlock avoidance mechanism depends on VCs monotonically decreasing. Any error that causes a VC to decrease in level is tolerable. Errors that bump the VC up can cause a deadlock.

But the deadlock is only an issue for packets that make it into the packet store. If a packet is bypassed from one node to another, then there was no buffer contention to cause a deadlock. Packets that are stored however, have the opportunity to negotiate their way into a deadly embrace.

For this reason, when we write a packet into a packet store, we examine the packet type. If the CRC is good and the packet type is poisoned, we immediately free the packet from the buffer.

This will not prevent a poisoned packet from traveling through the fabric, but it will prevent such a packet from locking up a packet store slot, which is the source of our potential deadlock.

3.8.3 Transient Bit Errors on the Link

Packets that are corrupted while traveling over the internode link will be resent by the upstream node. This is how.

Consider two nodes at either end of the link. U is the upstream node, transmitting data to D, the downstream node. Each packet sent by U carries a serial number (the LSN, or Last Sequence Number) that increments with each newly transmitted packet and is 4 bits wide. As U's output block for this link (OB) sends each packet, it will write the packet to a replay buffer. The replay buffer is indexed by LSN.

D, the downstream node, checks each packet as it arrives for errors. If a packet arrives without error, D loads the packet's LSN into the "Last Good Sequence Number" register in the link's input block (IB). At some time in the very near future, the current value of the Last Good Sequence Number will be sent back up the control link from D to U in a control packet.

When each control packet arrives at U, OB will examine two fields. If the Error bit in the first byte of the packet is clear, then the LGSN from the downstream node will be sent to the replay buffer. The replay buffer will release all packets up to and including the LGSN, as they have been acknowledged by the downstream node.

If D detects a CRC error, the IB will enter the Error Detected state and will ignore all incoming packets while in this state. The IB will set the "Error" bit in all outgoing control packets sent to the upstream node.

U will eventually receive a control packet (whose CRC checks out) that has the Error bit set. This tells U that all packets after the LGSN in that packet were ignored and must be resent. Before beginning the retransmission, U will send IDLE packets with a bit set in the IDLE FORD indicating that the Error is being acknowledged. The OB will then wait until it sees a control packet from D that has a clear Error bit.

The IB on node D will see at least one IDLE FORD with the error acknowledge bit set. IB will leave the "Error Detected" state, clear the Error bit in the first byte of outgoing control packets, and await resumption of the packet stream from U.

U will then receive a control packet that has the Error bit clear. This is the completion of the link error handshake. The OB on node U will begin sending packets out of its replay buffer beginning with the LSN after the LGSN that arrived in the most recent control packet. Once it has resent all the packets in its replay buffer, it will resume normal operation.

Note that packets are only freed from the replay buffer after U has received some positive acknowledgment from D via a control packet. The LGSN field tells U that all packets up to and including LGSN have been received correctly. The replay buffer can hold 16 packets, but in fact the OB stops transmitting if it contains 15 packets. The entry corresponding to the LGSN that arrived in the most recent control packet must not be used, or the

acknowledgment protocol becomes ambiguous. Example: If LSN3 was received last, and an OB sends data packets 4 through 15 and then 0 through 3, it can't tell if the next control packet acknowledging LSN3 has acknowledged 0 entries or all 16. To avoid this confusion, the OB would only send data packets 4 through 15 and 0 through 2 and then wait, avoiding LSN3 because it equals the LGSN.

When the replay buffer is filled, the OB inhibits global arbitration so that no more packets are sent out. Normally, the replay buffer should never fill up, as the round-trip latency from U to D and back again to U is short enough that slots in the OB will be freed up more quickly than they are consumed as long as there are no errors on the control link.[5]

If the retransmission fails, we'll keep attempting to retry. Retry events are counted and can cause an interrupt when the count exceeds a preset threshold. The link logic also maintains counts of framing errors and symbol translation errors. These are handled in the link control logic.

3.8.4 Corrupted VC

Either the VC got corrupted as part of a CRC error that we'll find when the EOP comes along, or it was corrupted at some previous stage and is the result of scrubbing the CRC for a poisoned packet that was generated by recovery mentioned above. (All packets get good CRC when they're sent out, even if they've been poisoned.)

Sooner or later, some bit error will corrupt a VC. There are two ways we can find that the VC has been corrupted.

1. We are supposed to decrement the VC and it is already 0. This is flagged as a VcDecrError and the packet is dropped by the input block. It would be dangerous to allow the packet to continue on to other nodes, because it has broken the VC decrement rule that allows the fabric to be deadlock-free. In this case the CRC is good, and we must NOT cause replay because the replayed packet would have the same problem.

2. The buffer index points to a buffer belonging to VC x (as opposed to the free pool) and the packet is traveling on VC y. In this case, we know the VC is broken and the CRC will not match. (the VC got corrupted on the wires.) Then we use the normal CRC mismatch recovery mechanism to ask for a retransmission. The IB will find the CRC mismatch. The packet store should free the buffer.

3.8.5 Corrupted Route

If the route was corrupted on the link, the corruption will cause a CRC mismatch, in which case a poisoned packet will be delivered to somebody – probably not the intended recipient. In this case the link will retry the transmission and a good – non-poisoned – packet will be resent. The retry packet will get to the ultimate destination. The poisoned packet will wander around for a while and either get delivered to some destination – where it will be discarded as a poisoned packet – or it will arrive at a node where the VC will be decremented from 0. In this latter case, the packet will be routed to the DMA engine as described in 3.8.4.

As in the case of a corrupted VC, the route could have been corrupted at an earlier stage or as the result of a flipped bit in the switch (*e.g.* error in the packet store). In this case, the packet carrying the corrupted route will be poisoned. In this case, the packet will wander around the network until it is delivered to some node – probably not the intended recipient – or is dropped because of an exhausted VC.

3.8.6 Corrupted Buffer Index

The packet store (within an XB block) may find that the buffer index of a packet points to a packet buffer entry that is already full. The packet store will ignore the packet – that is, it will not write the packet into a packet buffer entry. If the buffer index is corrupted such that it places the packet in an unused buffer, the buffer slot will still be freed, as the CRC will not match. Packets that arrive with a bad CRC will never occupy packet store space – at most, they will be deleted from the packet store as soon as the IB tells the XB that the CRC was bad. The IB will ask for a retransmission, since the CRC will not match.

[5] What is the worst case delay of acknowledgment, assuming no errors in control packets? A packet P1 is sent downstream that is 20 FORDs long. In the worst case, a control packet begins just as that packet is completing, so it is not acknowledged until the second control packet. Two control packets take 30 cycles. Add 3 cycles in each direction for latency of the link. The worst case delay is around 20+30+3+3=56 cycles, which is enough time for 14 minimum sized packets to be sent. So even in the worst case, the replay buffer should not fill up unless there are bit errors on the link.

3.8.7 Corrupted LSN

If the LSN field is corrupted in transit, the input block will discover that the CRC is bad. It doesn't trust the LSN field until the CRC is checked, so no special recovery mechanism is needed. This type of corruption will just cause the FswPktCrcError counter to increment.

3.8.8 Misc. Bad Data (CRC Mismatch)

In this case, the IB will detect a bad CRC on the incoming packet. It will change the packet type to Poison as it forwards the data to the XB and the OB. Also the IB asserts a BadPacket signal to the XB so that the packet can be discarded from a crosspoint buffer. The IB goes into replay so that the packet will be retransmitted.

3.8.9 Uncorrectable ECC Error in Packet Store or Replay Buffer

This is bad, since we'll end up with a non-delivered packet. When an uncorrectable ECC error is detected, the memory module asserts a double bit error flag which tells the OBX output mux to poison the packet. Also a CSR bit is set which, if enabled by software, will trigger an interrupt.

3.8.10 Uncorrectable ECC Error on Data to DMA Engine

When a crosspoint buffer detects an uncorrectable ECC error, it asserts a double bit error flag which tells the DMA output block to poison the packet. Also a CSR bit is set which, if enabled by software, will trigger an interrupt.

3.8.11 Uncorrectable ECC Error on Data from DMA Engine

The ICE9 memory system uses ECC to protect data from the moment it is written to an L2 segment until it reaches the fabric switch input block. For typical packet data, the processor generates the data in its L1 and asks the DMA to send it to a remote node. As that packet is sent onto the CSW to the DMA, an ECC code is generated that moves with the data as it goes through the CSW, DMA packet buffers, and into the fabric switch. In the fabric switch, the DMA input block corrects ECC errors before sending the data to a crosspoint buffer or to the output block for bypass. If there is a double bit error, it poisons the packet and asserts BadPacket at EoP time. (This is the same as what a normal input block does when it discovers any other kind of error.)

3.8.12 Upstream Link Goes Down

The fabric switch monitors the MissionMode signals coming from the 3 fabric link receivers to see if any upstream link has gone down. If an upstream link goes down, the fabric switch will treat any packets that are currently being received as error packets and enter replay. It will stop sending control packets upstream while MissionMode is down, and resume sending them after MissionMode goes back up.

A processor on the node can learn that the upstream link is down by an interrupt from the fabric link, or by polling the link CSRs.

3.8.13 Downstream Link Goes Down

The fabric switch monitors the MissionMode signals coming from the 3 fabric link transmitters to see if a downstream link has gone down. If a downstream link goes down, the fabric switch will stop sending any new packets to the corresponding FLT. Packets that have been sent out already, or are currently being sent out, will remain in the replay buffer. Any control packets coming from the downstream fabric switch will be ignored while MissionMode is deasserted. Eventually, the link will go back up, MissionMode will be asserted again, and the fabric switch will resume its usual output behavior: sending packets downstream as long as buffers are available and accepting good control packets. There is no mechanism for a processor to extract packets from switch buffers or to drop packets destined for a bad link. A processor on the node can learn that the downstream link is down by an interrupt from the fabric link, or by polling the link CSRs.

What happens to the system as a whole if a downstream link goes down forever? The fabric switch detects the loss of MissionMode from the FLT, and stops sending new packets on that link. Packets accumulate in the 4 crosspoint buffers that feed that output port, and eventually the buffers fill up. Control packets carry that information to the upstream switch, so its crosspoint buffers start to fill. In the upstream switch, the packets

intended for the downed link fill up its four crosspoint buffers, and other traffic (not routed through the downed link) gets stuck waiting for available buffers. The congestion propagates back through the fabric and eventually DMA engines stall because they can't put any more packets into the fabric. The fabric grinds to a halt.

On the positive side, all of this will resolve itself quickly after the affected link comes back online. But if the link is determined to be down permanently, what can we do? I will describe a way to recover, in part to demonstrate why we have decided not to attempt it in this version of the chip. Let's say that FLT2 on node X reports that the link is down, and software determines that the link is down for so long that it will never come back up. The fabric switch crosspoint buffers and replay buffers are full of packets destined for that link. First the processor would send a LinkDown message to every node, including itself, saying that it must must recompute its routing tables to avoid the affected link. Sending packets across the fabric as usual would not work, because parts of the fabric may be stuck by this time. Using the Out-of-Band channels or the system service processor would be possible. When a node Y receives the LinkDown message, it must send one LastPacketOnRoute packet along each route that touches the affected link, recompute its routing tables to avoid the downed link, then suspend sending any traffic along the affected routes until it gets a LastPacketOnRouteAck. Upon receiving LastPacketOnRouteAck, it can resume normal traffic along the new route. This handshake guarantees that all packets on the old route are delivered before any packets on the new route. After sending the LinkDown message, node X can start rerouting packets; it pulls them out of the replay buffer in order, generates a new header that routes the packet to the intended destination though a working link, and injects it into the fabric via its DMA. (NOTE: The FSW would need a mode that allows packets to flow into the replay buffer even though MissionMode is down.) Node X may have to do continue this for a very long time, until it drains the fabric and every DMA engine's queues of any packets that required this link. Eventually it starts to see LastPacketOnRoute messages, and sends LastPacketOnRouteAck messages back to the sender so that they can resume sending traffic normally on the new route. It may be possible to know exactly how many LastPacketOnRoute messages to expect so that node X knows when to stop rerouting packet, but it's probably easier to just do it indefinitely. A maskable interrupt that notifies the processor when the replay buffer is nonempty might be useful here.

Having said all of that, based on the complexity of recovering from link failures without dropping or reordering any packets, and the hardware, software, and verification work involved in making this possible, we have decided NOT to support this. For this version of the chip, our strategy is to hope that the link comes up again, and if it doesn't? Packets were lost in the switch, so do a machine check.

3.9 The Control/Status Register Path

3.10 Components and Hierarchy

3.10.1 Switch Top level

3.10.1.1 External Ports

Inputs

chaini_scbs_dat_sr Input chain for Serial Configuration Bus (SCB). All CSRs are accessed through the SCB.

flrX_fsw_InDat_s0a<63:0> Input data from port X, where X is 0,1,2

flrX_fsw_DatVal_s0a True if InDat is carrying valid data.

flrX_fsw_SoP_s0a True if InDat is carrying the first FORD in a packet – (Start-of-Packet)

flrX_fsw_EoP_s0a True if InDat is carrying the last FORD in a packet – (End-of-Packet)

flrX_fsw_Idle_s0a True if this is an inter-packet IDLE FORD – this carries out-of-band and error control information.

flrX_fsw_MissionMode When clear, the fabric switch must ignore the SoP, EoP, Idle, DatVal, and InDat signals coming from Fabric Link Receiver X. When set, the signals from Fabric Link Receiver X are valid.

fltX_fsw_CtlDat_s0a<7:0> Flow control, error notification, and out-of-band information from port X's downstream node.

fltX_fsw_NewCtlPkt_s0a CtlDat should be ignored, the next cycle's value will be the first byte in a flow control packet coming from transmit port X's downstream node.

fltX_fsw_CtlEoP_s0a The byte carried by CtlDat is the last in this control packet.

fltX_fsw_MissionMode_s0a When clear, the fabric switch must not assert fsw_fltX_DatVal_s2a, and it must ignore any control packet traffic from Fabric Link Transmitter X. After MissionMode goes up, the fabric switch must not send any data packet until after a good control packet has been received.

dma_fsw_InDatX_s0a<71:0> Data from the DMA engine destined for output port X. Bits 63:0 are the data, and bits 71:64 are a 64-bit ECC on the data.

dma_fsw_DatValX_s0a Corresponding InDatX is valid

dma_fsw_SoPX_s0a Corresponding InDatX is the first FORD in a packet

dma_fsw_EoPX_s0a Corresponding InDatX is the last FORD in a packet

dma_fsw_RdyX_s1a Port X in the DMA engine is ready for a new packet from switch input port X.

Outputs

scbs_chaino_dat_sr Output chain for Serial Configuration Bus (SCB). All CSRs are accessed through the SCB.

fsw_xxx_Int_sa Active-high interrupt triggered when any bit in the Interrupt Cause Register which is not masked by the Interrupt Mask register is set. The processor must determine the exact interrupt cause by reading CSRs.

fsw_fltX_OutDat_s2a<63:0> Output data to the fabric link transmitter for port X

fsw_fltX_DatVal_s2a Corresponding OutDat is worth looking at

fsw_fltX_SoP_s2a Corresponding OutDat is the first FORD in a packet

fsw_fltX_EoP_s2a Corresponding OutDat is the last FORD in a packet

fsw_fltX_Idle_s2a Corresponding OutDat carries out-of-band and error control information

fsw_flrX_CtlDat_s3a<7:0> Flow control data for the upstream control link from receive port X

fsw_flrX_NewCtlPkt_s3a Corresponding CtlDat should be ignored, next value is the first data in a control packet.

fsw_flrX_CtlEoP_s3a Corresponding CtlDat should be ignored, this is the last byte in a control packet.

fsw_dma_OutDatX_s2a<71:0> Output data from switch input port X to the receive port buffer X in the DMA engine. Bits 63:0 are the data, and bits 71:64 are a 64-bit ECC on the data. The ECC protects against single bit errors in DMA memories, DDR, and the L2 cache.

fsw_dma_DatValX_s2a True if corresponding OutDat is worth looking at

fsw_dma_SoPX_s2a You've probably noticed a pattern by now

fsw_dma_EoPX_s2a Corresponding OutDat is the last FORD in a packet

fsw_dma_BufAvailX_s3a If true, the DMA engine may send a transmit packet from DMA engine transmit buffer X to switch port X.

3.10.1.2 Serial Configuration Bus Interface

The fabric switch's control/status registers are accessible through the SCB (Serial Configuration Bus) interface. To connect to the SCB, a module must simply instantiate an SCB slave module, and connect it to a global SCB chain. The input is connected to chaini_scbs_dat_sr and the output is connected to scbs_chaino_dat_sr.

The SCB bus and the SCB slave module are documented in 10 (the Serial Configuration Bus chapter).

The FSW's control/status registers are documented in section 3.12.5.

3.10.1.3 Interrupt Outputs

The fabric switch produces an interrupt signal, when certain kinds of errors are detected or when out-of-band flags toggle. The interrupts are sent to the CSW, which distributes them to processors appropriately. The interrupt outputs are level sensitive, active-high signals. Interrupts turn on when the condition is first detected, and remain on until cleared via the SCB.

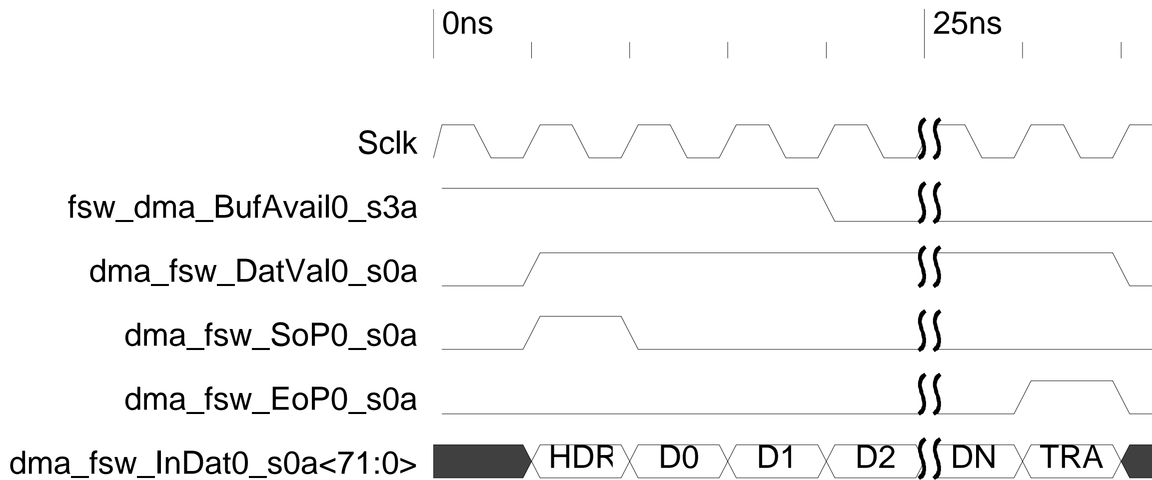


Figure 3.2: Data Path from DMA Engine Transmit Port 0 to Fabric Switch

3.10.1.4 The DMA to Fabric Switch Interface

Transmit Data Path

dma_fsw_InDatX_s0a<71:0> Data+ECC from the DMA engine destined for output port X

dma_fsw_DatValX_s0a Corresponding InDatX is valid. Asserted only during all data packet cycles including the SoP and EoP.

dma_fsw_SoPX_s0a Corresponding InDatX is the first FORD in a packet

dma_fsw_EoPX_s0a Corresponding InDatX is the last FORD in a packet

fsw_dma_BufAvailX_s3a If true, the DMA engine may send a transmit packet from DMA engine transmit buffer X to switch port X.

The DMA input block (DMAI) asserts fsw_dma_BufAvailX_s3a when it has space for at least two packets in the outgoing crosspoint buffer. A few cycles after reset, BufAvail is asserted because all crosspoint buffers are free. Afterwards, if a newly arriving packet consumes the next-to-last buffer entry, the DMAI deasserts BufAvail within five Sclock cycles of the assertion of dma_fsw_SopN_s0a. When two or more buffers become free, the DMAI asserts BufAvail again. Deassertion of BufAvail is always the result of a packet coming in from the DMA, but assertion of BufAvail can happen at any time.

The minimum sized packet is four FORDs: two payload FORDS, plus the head and tail FORDs. The maximum sized packet is twenty FORDs, of which eighteen form the payload.

No retries are ever required on this interface. All packets are assumed to arrive in good health. Single bit ECC errors will be corrected on the fly before the data enters the XBX.

SoP and EoP are each asserted for exactly one Sclock cycle. The two are always paired, with exactly one EoP assertion for every assertion of SoP.

The format of the header and trailer FORDs is described in Sections 3.4.1.1 and 3.4.1.2. In the header FORD, the DMA engine fills in Vc, NumFords, HasCtrl, and Route. The FSW output block fills in XbeTarget and Lsn, and the link fills in the SoP. In the trailer FORD, the DMA engine fills in Type, ProcessIndex, and UnixProcessId, and sets Crc32 to zero. The FSW output block fills in Crc32, and the link fills in EoP as the packet goes onto the wire.

Note that each DMA input block is connected to exactly one output block, so there is no mystery about which output port the packet will leave on. The route field is NOT shifted in the DMA input block. The two LSBs of route represent the output port number in the downstream switch. The two LSBs are used in the crosspoint buffer while arbitrating and selecting a downstream XbeTarget.

Receive Data Path

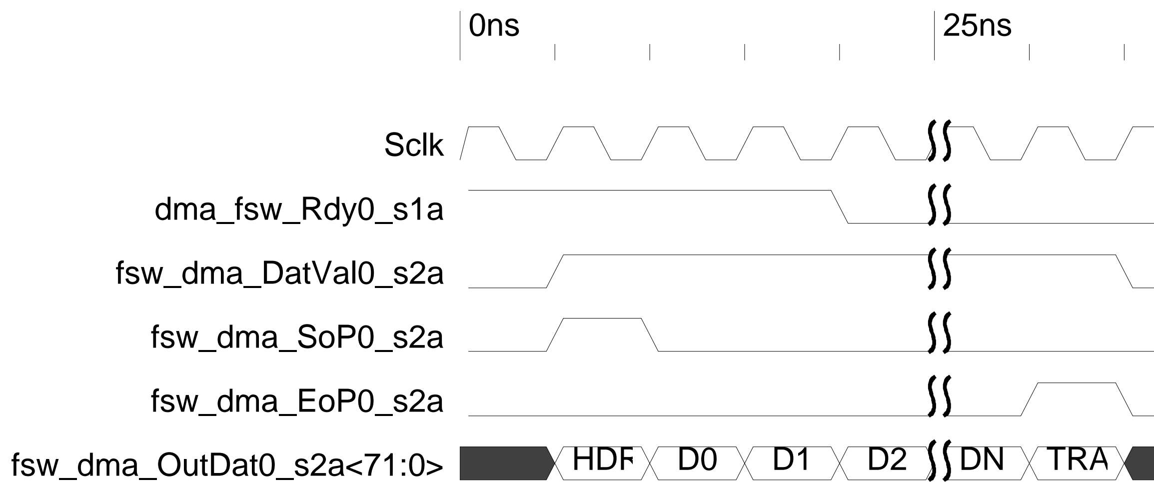


Figure 3.3: Data Path from Fabric Switch to DMA Engine Receive Port 0

fsw_dma_OutDatX_s2a<71:0> Output data+ECC from switch input port X to the receive port buffer X in the DMA engine. The DMAO module generates the ECC code on the fly as the packet travels from a crosspoint buffer to the DMA engine.

fsw_dma_DatValX_s2a True if corresponding OutDat is worth looking at. Asserted only during all data packet cycles including the SoP and EoP.

fsw_dma_SoPX_s2a You've probably noticed a pattern by now

fsw_dma_EoPX_s2a Corresponding OutDat is the last FORD in a packet

dma_fsw_RdyX_s1a Port X in the DMA engine is ready for a new packet from switch input port X.

The DMA engine asserts dma_fsw_RdyX_s1a whenever it has space available in its port X receive buffer. If the packet sent from the switch to the DMA engine consumes the last such buffer, the DMA engine must de-assert RdyX within no later than 3 Sclock cycles after the assertion of fsw_dma_SoPX_s2a.

The switch asserts DatVal0, and SoP0 drives the header FORD onto OutDat0, followed by the payload (of no fewer than 2 payload FORDS, and no more than 18 payload FORDs) and the tail FORD.

Data transfer along this path is assumed perfect. There is no recalculation of CRC, replay logic, length checking, etc. The only reason ECC is there is to protect the data later on, in DMA memories and beyond.

SoP and EoP are each asserted for exactly one Sclock cycle. The two are always paired, with exactly one EoP assertion for every assertion of SoP.

There is a potential race in this interface, in which the FSW consumes the last available buffer in the DMA, the DMA deasserts dma_fsw_RdyN, but the FSW doesn't hear in time to suppress the next packet. To avoid this race, the DMA output block will observe the following rule: it will never assert SoP sooner than 6 sclk cycles after the previous SoP. This means that very short packets will have a gap after them. Long packets are not affected.

The format of the header and tail FORDs is described Sections 3.4.1.1 and 3.4.1.2.

3.10.1.5 The Fabric Link Receiver (FLR) to Switch Interface

Receive Data Path

flrX_fsw_InDat_s0a<63:0> Data arriving through link receiver port X

flrX_fsw_DatVal_s0a If true, then InDat is worth looking at

flrX_fsw_Idle_s0a If true, then InDat is carrying IDLE FORD information (error control and status)

flrX_fsw_SoP_s0a If true, then InDat is the first FORD of a packet

flrX_fsw_EoP_s0a If true, then InDat is the last FORD of a packet

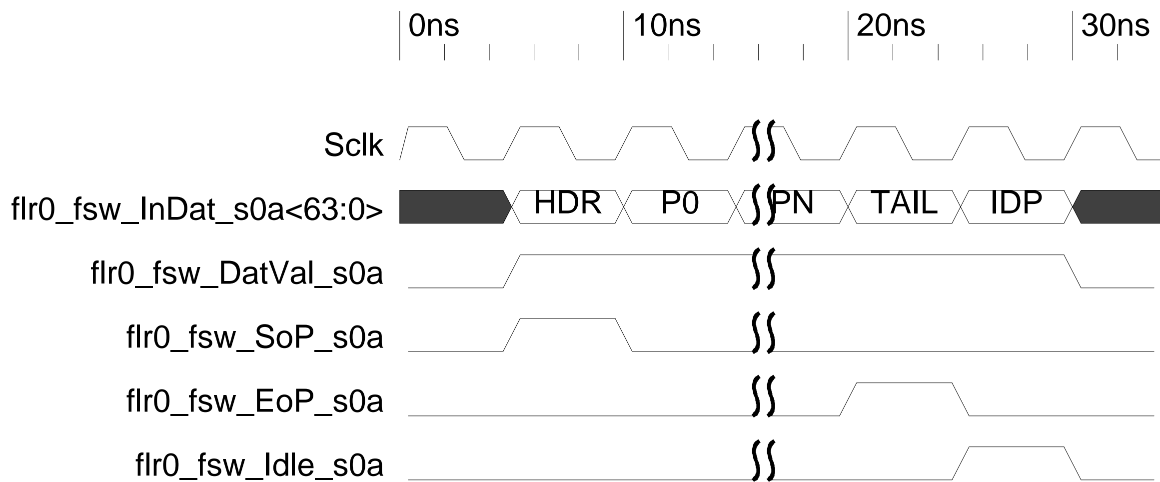


Figure 3.4: Receive Port to Fabric Switch Data Path
Not shown: flt0_fsw_MissionMode_s0a is asserted throughout.
(P0...PN are FORDs 0 through N of the payload. $4 \leq N \leq 18$.See Figure 3.4.1.3.
IDP is an IDLE packet FORD. See Section 3.4.1.4.)

The fabric link receivers (FLR0, FLR1, FLR2) send data (flrX_fsw_InDat_s0a<63:0>) and associated control information to each of the corresponding input blocks in the fabric switch. Figure 3.6 shows the relative timing between the control signals and the data. The first FORD is always marked by the presence of the SoP signal, and the last is marked by EoP. All signals are ignored if DatVal is not asserted.

Note that there must always be exactly one cycle of EoP to follow ever SoP. SoP and EoP should never be asserted for more than one cycle.

The switch also receives control and status information via IDLE packets. These are identified by the simultaneous assertion of both DatVal and Idle. Section 3.4.1.4 shows the format of this packet.

Receive Control Packet Path

fsw_flrX_DatVal_s3a The data on fsw_flrX_CtlDat_s3a is valid

fsw_flrX_CtlDat_s3a<7:0> One byte of information to be sent to the upstream node via receiver port X's control link output

fsw_flrX_NewCtlPkt_s3a If true, then ignore CtlDat, this is the start of a control packet. (Next cycle's CtlDat will be the first payload byte in the control packet

Each downstream node sends flow control and error information back to the upstream node via the control link through the appropriate fabric link receiver. Control Packets are 15 bytes long including the SOP symbol that delimits packets. The packet is covered by a 32 bit CRC. See Section 3.4.2.1 for a description of the packet format.

NewCtlPkt may be asserted for more than one cycle at a time, in this case the start of the next control packet's payload is delayed until the deassertion of NewCtlPkt.

The important parts of the control packet regulate both error recovery and buffer allocation.

To review, buffer allocation is performed in the upstream switch. Each time a packet is transmitted by an upstream switch, it is assigned a slot in the downstream node's packet store within the appropriate Crosspoint Buffer (XB) along with a packet sequence number (LSN). The upstream node remembers that buffer B was consumed by the packet with LSN L in his own LocalBufferBusy mask for the destination output port on the downstream node.[6] The upstream node will never assign a packet to a buffer it knows to be in use. The upstream node assumes a buffer is in use if it is assigned according to the LocalBufferBusy mask, or if it is assigned according to the PxBusy field in the last received control packet (where 'x' is the output port number.) The upstream node clears packet L's buffer busy bit in the LocalBufferBusy mask when the LSN reported in the last received control packet is equal to or greater than L.

[6]Note that the buffer assignment is for a particular crosspoint buffer, so there is a LocalBufferBusy mask for each of the four output ports on the downstream node. Similarly, the downstream node reports buffer busy status for each of its four output ports.

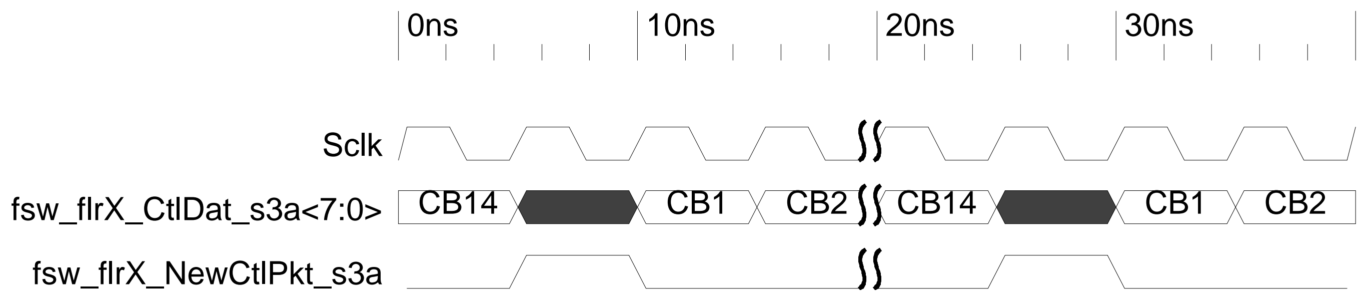


Figure 3.5: Fabric Switch to Receive Port Control Data Path
(CB0...CB14 are bytes 0 through 14 in the control packet. See Figure 3.1.)
This picture doesn't show fsw_flrX_DatVal_s3a and fsw_flrX_MissionMode_s0a, both of which must be asserted during all 15 bytes of valid Control Packets.

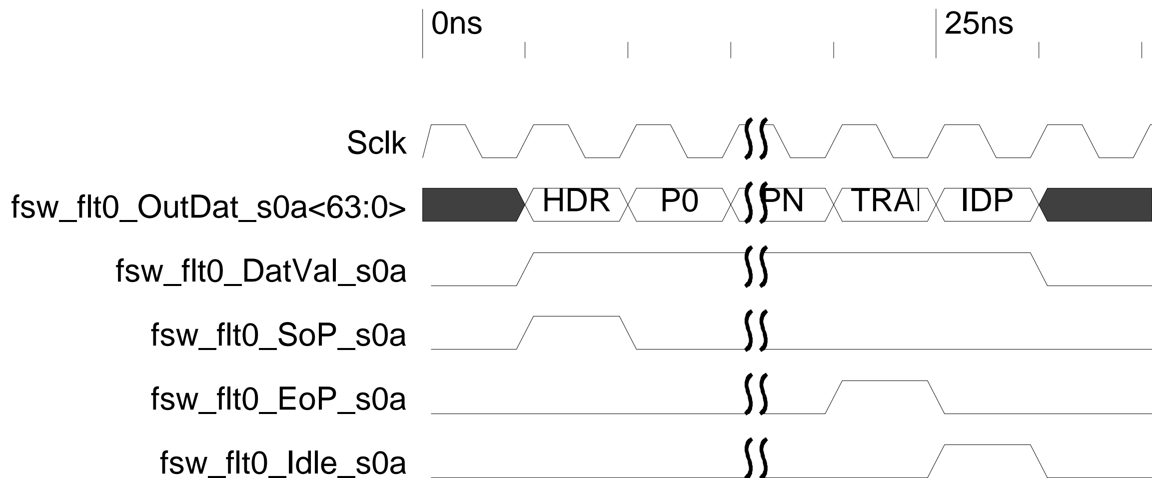


Figure 3.6: Fabric Switch to Transmit Port 0 Data Path
(P0...PN are FORDs 0 through N of the payload. $4 \leq N \leq 18$.See Figure 3.4.1.3.
IDP is an IDLE packet FORD. See Section 3.4.1.4.)

So, this is worth checking. The upstream node should never send a packet that is destined for a buffer that it should believe is busy. If such an event does occur, the packet will be ignored.

3.10.1.6 The Fabric Link Transmitter (FLT) to Switch Interface

Transmit Data Packet Path

fsw_fltX_OutDat_s2a<63:0> Output data from the switch to the downstream node

fsw_fltX_DatVal_s2a When true, OutDat is worth looking at

fsw_fltX_SoP_s2a When true, OutDat is the first FORD in a transmitted packet

fsw_fltX_EoP_s2a When true, OutDat is the last FORD in a transmitted packet

fsw_fltX_Idle_s2a When true, OutDat is carrying an IDLE FORD.

The transmit data packet path is the complement of the receive data packet path. Relative timing and meaning of the five signals is identical for both. Figure 3.6 shows the relative timing between the control signals and the data. Section 3.4.1.4 shows the format of this packet.

Transmit Control Packet Path

fltX_fsw_DatVal_s0a the data on fltX_fsw_CtlDat_s0a is valid

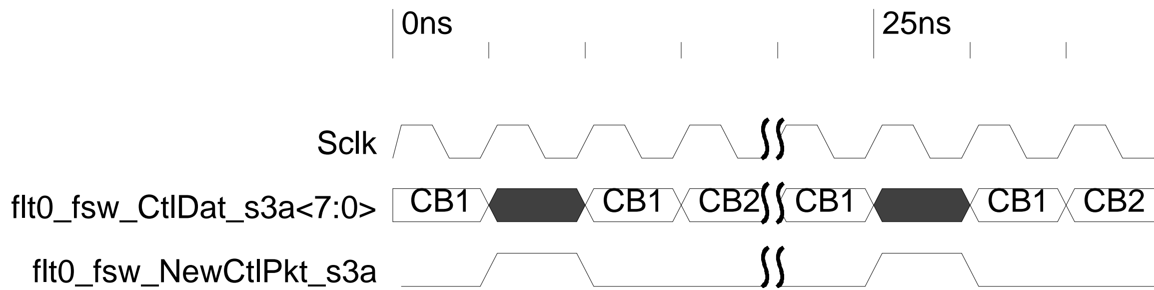


Figure 3.7: Transmit Port 0 to Fabric Switch Control Data Path
(CB0...CB14 are bytes 0 through 14 in the control packet. See Figure 3.1.)
This picture doesn't show flt0_fsw_DatVal_s3a, which must be asserted during all 15 bytes of valid Control Packets.

fltX_fsw_CtlDat_s0a another payload byte in a control packet

fltX_fsw_NewCtlPkt_s0a When true, ignore CtlDat, the next cycle's value will be the first byte in a new control packet. Also indicates the previous byte was the last in a control packet.

The transmit control packet path is the complement of the recieve control packet path. Control packets from the downstream node arrive at the fabric link transmit block and are forwarded to the output block (OB) in the switch. The OB parses the control packet (See Section 3.4.2.1) to determine the state of buffer allocation in the downstream node and to find the latest accepted packet sequence number. The timing and behavior of the two signals in this path are described in 3.7.

3.10.2 Interblock Signals

Figure 3.8 shows the signals between blocks of the fabric switch.

3.10.3 The Input Block

The input block (IB) distributes incoming FORDs from the attached input port to one the four crosspoint buffers (XBs) based on the routing field in the packet's first FORD. The IB also decrements the VC if necessary and performs CRC checking on the packet as its last FORD passes through. It also checks to detect packets that have been poisoned. Such packets are removed from the packet store in the XB soon after the last FORD has been written.

The IB remaps the virtual channel field in the header of each incoming packet based on the deadlock avoidance routing rules. It also shifts the routing vector two places to the right, by throwing away the two LSBs and shifting the input block number (0, 1, or 2) into the two MSBs. It uses the DecrVC register for this IB. DecrVC is a three bit vector, written by the SCB interface. If bit X in the vector for IB Y is set, then all packets arriving on port Y and destined for port X will have their VC decremented by one. Otherwise the VC field in the packet is unchanged.

Finally, the IB checks the CRC at the end of the incoming packet and signals any detected error back to the input port and forward to the crosspoint buffers. The CRC field is 32 bits wide and is contained in the last FORD in the packet. (See Section 3.8.1.)

The IB is also responsible for passing the "free buffer" vector from the arbitration array and the last good sequence number up to the input port. The IB builds control packets and sends them continuously to the upstream node. (See 3.4.2.)

3.10.3.1 Error Detection and Recovery Table

The following checks are performed on input data packets. Each of these checks becomes a column in the error behavior table below.

- Pro: Was there a protocol error? Check that SoP was always followed by EoP, and EoP was always followed by SoP, and they were never asserted in the same cycle. If so, set IbProtocolErr.
- DV: Was the flrN_fsw_DatVal signal ever low during the data packet? If so, set IbMissingDatavalid.

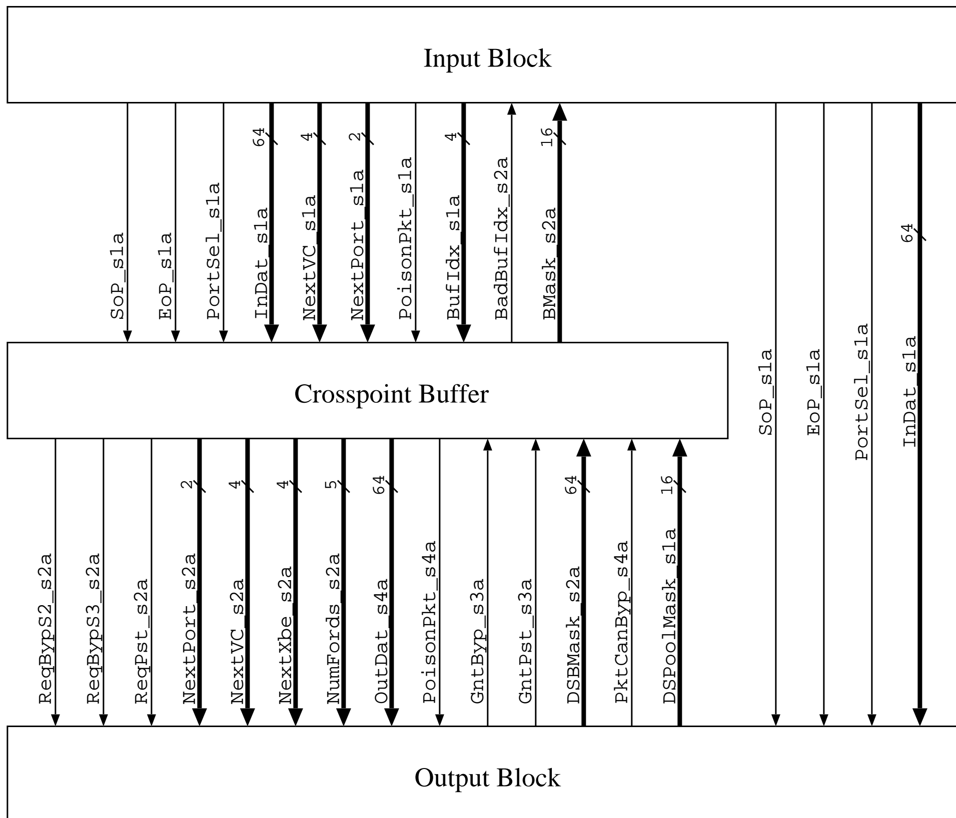


Figure 3.8: Interblock Signal Connections

- BNF: Bad NumFords. Was the NumFords field less than FSW_MINFORDS_PACKET or greater than FSW_MAXFORDS_PACKET? If so, set IbBadNumFords.
- Lsn: The LSN should always equal last good LSN plus one (with wrap around at 16). Was the LSN ever something other than the expected value? If so, set IbMissingLsn.
- LMin: Was the observed packet length less than the NumFords field specified? If so, set IbLengthErrMin.
- LMax: Was the observed packet length greater than the NumFords field specified? If so, set IbLengthErrMax.
- Xbe: Does the XB already have a packet in the buffer that the packet specified in XbeTarget? If so, set IbBadXbeTargetErr.
- Vc: Did the VC decrement below zero? If so, set IbVcDecrErr.
- Crc: Was there a CRC mismatch on a data packet? If so, increment R_FswDataCrcCounter (once per packet).

Based on the result of each of these checks, the input block may decide to

- Drop: Don't send the packet to the XB or OB, and increment FswPktPoisonCounter. (Dropping of an errored packet is only possible when error is visible from just the header.)
- Poison: Change the packet type to FSW_POISON_TYPE, and increment FswPktPoisonCounter.
- Replay: Start replay sequence to ask the upstream node to try again.
- Send: Send the packet normally and increment R_FswPktCounter.

The input block behaviors correspond to the different error checks according to the following table. The columns on the left are all the types of error checks, and a 1 means that the error was detected. The columns on the right are the action that the fabric switch will perform.

Pro	DV	BNF	Lsn	LMin	LMax	Xbe	Vc	Crc	Drop	Poison	Replay	Send
1	x	x	x	x	x	x	x	x	0	1	1	0
0	1	x	x	x	x	x	x	x	0	1	1	0
0	0	1	x	x	x	x	x	x	1	n/a	1	0
0	0	0	1	x	x	x	x	x	1	n/a	1	0
0	0	0	0	1	x	x	x	x	0	1	1	0
0	0	0	0	0	1	x	x	x	0	1	1	0
0	0	0	0	0	0	1	x	x	0	1	1	0
0	0	0	0	0	0	0	1	0	1	n/a	0	0
0	0	0	0	0	0	0	1	1	1	n/a	1	0
0	0	0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1

3.10.4 The Output Block

The output block performs global arbitration between the four attached crosspoint buffers, maintains a "replay buffer," modifies outgoing packets with updated sub-band and error handling information, and computes the new CRC. The output block stores data for all bypass paths, and for the 3-cycle bypass path the OB decides whether to allow the bypass or not.

Global arbitration is described in section 3.7.4.

The replay buffer holds up to 15 data packets that were recently sent over the outbound link. Every packet is recorded in the replay buffer as it is transmitted, and packets are deleted when they are acknowledged by the downstream switch. Each packet in the replay buffer is stored at a fixed address according to its link sequence number (assigned by the output block). When an error is signaled on the return control link, the output block stops sending packets from the crosspoint buffers and sources packets from the replay buffer instead.

3.10.5 The DMA Input Block

3.10.5.1 Error Detection and Recovery Table

The following checks are performed on data packets from the DMA. Each of these checks becomes a column in the error behavior table below.

- Ecc2Head: Was there a double-bit error in the header of the data packet? If so, set DmaiDoubleBitErr.
- BNF: Was the NumFords field out of range? If so, set DmaiBadNumFords. (NOTE: Check the NumFords field after ECC correction.)
- Ecc2Other: Was there a double-bit error in any ford other than the header? If so, set DmaiDoubleBitErr.
- Ecc1: Was there a single-bit error in any of the fords of the data packet? If so, set DmaiSingleBitErr.

Based on the result of each of these checks, the input block may decide to

- Drop: Don't send the packet to the XB or OB. (This is only possible when error is visible from just the header.)
- Poison: change the packet type to FSW_POISON_TYPE.
- Send: Send the packet normally and increment R_FswDmaiPktCounter.

The DMA input block behaviors correspond to the different error checks according to the following table. The columns on the left are all the types of error checks, and a 1 means that the error was detected. The columns on the right are the action that the block will perform. The CSR column gives the name of the flag that is set or the counter that is incremented.

Ecc2Head	BNF	Ecc2Other	Ecc1	Drop	Poison	Send
1	x	x	x	1	n/a	0
0	1	x	x	1	n/a	0
0	0	1	x	0	1	0
0	0	0	1	0	0	1
0	0	0	0	0	0	1

3.10.6 The DMA Output Block

3.10.7 The Crosspoint Buffer

3.10.7.1 The Arbitration Array

What does it need to do?

1. Maintain the "buffer busy" bits – set the bits as packets arrive, and clear them as they leave.
2. Remap virtual channel id's into downstream buffer requirements.
3. Send flow control information back to the input block.
4. Accept flow control information from the output block.
5. Register incoming packets for arbitration.
6. Fire arbitration when appropriate.
7. Launch packets from the crosspoint buffer into the output block.

The arbitration array selects the next XBE to be sent out of the XB based on which buffers are available downstream. First the XBE equal to the VC is considered; if it's available the next XBE will be equal to the VC number. Then all XBE entries whose bit is set in the PoolMask are considered, lowest order bits first. This XBE selection is part of the local arbitration stage. The arbitration array also keeps track of occupied entries in the XB and sends updates to the appropriate Input Block (IB) which will then pass the information upstream.

Upon arrival, each packet specifies the destination XBE in its first FORD. The destination is checked against the packet store's valid bits. If there's already a packet in that entry, the packet is ignored and treated as a BadXbeTargetError. Otherwise, the virtual channel specification from the first FORD is decoded into a 16 bit vector and ORed with PoolMask and stored for the destination XBE. The destination port on the downstream node (that is, the next routing token from the FORD) is written into the **EntNextPort_s2a<1:0>** register for the XBE. At the same time, the entry's age vector **XbeAgeVec_s2a<15:0>** is set to all 1's except for the bit in the age vector corresponding to the destination XBE.

The arriving packet will begin to participate in local arbitration in the following cycle if it's eligible.

The OB sends each arbitration array updated buffer busy masks for each of the four outbound ports on the destination node.

3.10.7.2 The Packet Store

The packet store (PS) is a 72 bit by 320 word RAM organized as 16 blocks of 20 words each. Each word consists of 64 bits of data protected by 8 bits of ECC. Each 20 word block comprises a crosspoint buffer entry (XBE). The PS gets its input data directly from the input unit (though the ECC is generated in the packet store) and the write address comes from the arbitration array. Similarly, the arbitration array sends the read address to the PS. The PS can simultaneously read and write. It runs off of the fabric switch clock (SClock) – nominally 200MHz.

For each XB entry, the packet store also stores a Valid bit, the VC (4 bits) and NextPort (2 bits) in flops, since those are needed in order to determine eligibility and make requests in every cycle.

3.11 Pipeline Timing

The following tables describe the pipeline stage in which different events occur. A brief version is presented first, followed by tables with more detailed descriptions.

3.11.1 Summary

Cycle	Description
S0	SoP data arrives from Fabric Link Receiver
S1	IBX checks for errors, modifies header, sends to XBX. XBX stores it.
S2	XBX chooses oldest eligible packet and asserts ReqPst to OBX. XBX starts to read packet store address 0 for the packet for which it is requesting. OBX does global arb, then per-VC-and-NextPort arb to select winning packet.
S3	OBX asserts GntPst to winning XBX, which starts to read packet store address 1 of winning packet.
S4	XBX delivers packet SoP to OBX. OBX fills LSN and CRC fields and flops data.
S5	OBX drives SoP data to Fabric Link Transmitter. All outputs are also stored in Replay Buffer.

3.11.2 Incoming Packet is Stored in Crosspoint Buffer, Arbitrates, and Wins

Each of these tables tells a story of how an input stimulus triggers actions within the block, and how quickly each block reacts to the stimulus. The first story follows a packet along a common path through the switch. It does not qualify for bypass, so it is stored in a crosspoint buffer until it wins arbitration and gets sent out. Because this is a common path that touches almost every subblock of the switch, we have used it to define the cycle numbers S0 through S5. Other tables will refer back to these cycle numbers as we describe faster and slower paths through the switch.

Cycle	Description
S0	The first ford of a packet (identified by the SoP pulse) is driven from the Fabric Link Receiver to the Fabric Switch IBX. The IBX performs minimal signal cleanup and then flops the data.
S1	The IBX checks the packet for errors (CRC, length, VC, and more). The routing string in the header is shifted, the VC is decremented. The IBX forwards packet data to four crosspoint buffers in mid-S1, along with ibx_xbx_PortSel wires which tell which crosspoint buffer is selected. The IBX also forwards packet data to every OBX to support bypass, but bypass will not be covered in this table. In the selected XB, compute the new age vectors and set the valid bit for the XB entry which will accept the packet. The packet store computes ECC and begins to write the packet on the S2 edge. Or, if the XB already has a packet in that buffer entry, prepare to raise xbx_ibx_BadBufIdx in S2.
S2	In the XB, the new age vector and valid bit causes the packet to participate in local arbitration. If a free buffer is available, the packet is eligible; if it is the oldest eligible, it wins local arbitration. If any packet is eligible, the XB asserts xbx_obx_ReqPst, xbx_obx_NextVC, and xbx_obx_NextPort, and it starts to read address 0 of the packet. Unless global arbitration is inhibited, the OBX performs global arb betwen the four XBs that drive it. It takes the VC and NextPort of the winner and arbitrates between any of the four candidate packets that have the same VC and NextPort as the winner. The result of arbitration is flopped in OBX. Meanwhile, the set of XBE valid bits are sent from XBX to IBX (xbx_ibx_BMask_s2a) so that it can forward them to the upstream switch in control packets.
S3	OBX asserts obx_xbx_GntPst signal to the winning XB. The winning packet is whichever packet caused the XB to request in the previous cycle. (Since local arb happens constantly based on buffer busy masks, it is conceivable that the local arb winner in S3 is different from the winner in S2.) The XB starts to read address 1 of the packet. As soon as the grant arrives, the XB entry becomes free. The busy mask sent from XB to IBX reflects this in S4. This sounds scary, but it is safe because we are now committed to reading the XBE one ford per cycle; no incoming packet could ever overtake the read. We must be careful with how we represent the packet length so that it's not overwritten if a new packet takes this XBE during the read.
S4	In the XB, the data from address 0 is now available. The XB does ECC correction and its output mux selects the packet store output and sends it to the OBX. The XB is responsible for filling in the "next XB" entry field in the header as it goes out to OBX. In OBX, the output mux selects data from the winning XB, inserts an LSN, computes CRC, and the data is flopped. As the packet header passes through, the length field is captured and used to decide how long to inhibit arbitration for the next packet. For a minimum length packet of 4 fords, arb can begin again in S6 so that it completes just in time to chose the next packet. As the length increases by one, the arb inhibit window increases by one as well. Arb inhibit is tuned so that a winner is chosen just in time to be sent immediately after the previous packet ends. The winning packet will consume a buffer in the downstream switch, and the OBX must remember that fact in a "pessimistic busy mask" until a control packet arrives that acknowledges this packet. The pessimistic busy mask is updated so that in S4, the busy masks sent to the XBs reflect buffer status after accounting for the winner selected in S2.
S5	Data comes out of the flop and is sent to Fabric Link Transmitter. At the same time it is also written into the Replay Buffer, in case the packet needs to be resent later. The replay buffer recomputes ECC and stores 72 bits of data. If the packet being sent is only 4 fords long, and any XBs are requesting, global arbitration can begin in S6.

3.11.3 Packet Must Wait for Available Downstream Buffer

A packet arrives in a crosspoint buffer, but it cannot arbitrate and be sent out the output port because it is waiting for a downstream buffer to become free. Let's assume that we're not waiting for a pessimistic bit in the local busy mask; we're just waiting for the downstream buffer.

Cycle	Description
S0	SoP arrives from Fabric Link Receiver
S1	IBX checks for errors, modifies header, sends to XBX. XBX stores it.
Stall 1	The packet requires a downstream buffer that is not available, so it cannot participate in local arb. Other packets that need different buffers will continue to flow.
...	Packet gets bored and falls asleep.
Stall N-1	The end of a control packet arrives (from the FLT) which says that the required buffer has been freed. Control packet data is flopped in the OBX.
Stall N	OBX checks the CRC and decides that the control packet is valid. OBX sends updated busy mask to each XBX. XBX flops the busy mask.
S2	Now the packet is eligible and (if it's the oldest eligible) wins local arb. The XB asserts ReqPst to OBX. OBX does global arb, then per-VC-and-NextPort arb to select winning packet. XB begins to read packet at address 0.
S3	OBX asserts GntPst to winning XBX, which starts to read packet at address 1.
S4	XBX delivers packet SoP to OBX. OBX fills LSN and CRC fields and flops data.
S5	OBX drives data to Fabric Link Transmitter. All outputs are also stored in Replay Buffer.

3.11.4 Packet Loses Global Arb, but Wins on Second Try

Cycle	Description
S0	SoP data arrives from Fabric Link Receiver on IB1 destined for OB2.
S1	IB1 checks for errors, modifies header, sends to XB12. XB12 stores it.
S2	Now the packet is eligible and (if it's the oldest eligible) wins local arb. The XB asserts ReqPst to OB2. OB2 does global arb, then per-VC-and-NextPort arb to select winning packet. XB12 begins to read packet at address 0. But OB2 selects a packet from XB02 that is 8 fords long instead. Global arb is disabled for 8 cycles.
Stall 1	Packet is still oldest eligible, so XB12 continues to assert ReqPst. But global arb is disabled so the request is ignored. XB12 reads the packet at address 0 again.
...	
Stall 7	Packet is still oldest eligible, so XB12 continues to assert ReqPst. But global arb is disabled so the request is ignored. XB12 reads the packet at address 0 again.
Stall 8	Global arb is enabled again. This time global arb selects the packet in XB12.
S3	OB2 asserts GntPst to XB12, which finally starts to read address 1 of the packet. XB12 invalidates the crosspoint buffer entry.
S4	XB12 delivers packet SoP to OB2. OB2 fills LSN and CRC fields and flops data.
S5	OB2 drives data to Fabric Link Transmitter. All outputs are also stored in Replay Buffer.

3.11.5 Packet with CRC Error is Poisoned and Sent Anyway

Because we do cut-through routing, a packet may already be on its way out to the next node before we discover an error such as CRC, which is only detectable at the end. All we can do is poison the packet (change the type field in the last ford) and try to cancel it if it's still waiting to be transmitted. In this example the packet is 6 fords long: FORD1 through FORD6.

Cycle	Description
S0	FORD1 arrives from Fabric Link Receiver.
S1	IBX checks for errors, modifies header, sends to XBX. XBX stores FORD1. FORD2 arrives from FLR, and input block continues to compute CRC.
S2	XBX chooses this packet in local arbitration and asserts ReqPst. OBX does global arb and selects this packet.
S3	OBX asserts GntPst to the winning XBX. XBX clears the valid bit for the XB entry so that it can be reused.
S4	XBX delivers FORD1 to OBX. OBX fills LSN and flops data.
S5	OBX drives FORD1 to Fabric Link Transmitter. FORD6 arrives from FLR. The input block computes final CRC and it doesn't match the CRC field in FORD6. Now we know there was a CRC error, but the XB and OBX have already begun to send the packet. The IB changes the packet type to Poison as it sends to XB, asserts ibx_xbx_BadPacket_s1a, and increments a CRC error count register. The BadPacket signal causes the XB to clear the valid bit for the XB entry, but it was already cleared by GntPst in S3 so this has no effect.
S6	XBX sends FORD3 to OBX. OBX drives FORD2 to FLT.
S7	XBX sends FORD4 to OBX. OBX drives FORD3 to FLT.
S8	XBX sends FORD5 to OBX. OBX drives FORD4 to FLT.
S9	XBX sends FORD6 to OBX. (It already has the Poison type because the IB changed it.) OBX drives FORD5 to FLT.
S10	OBX drives FORD6 to FLT.

3.11.6 Packet with CRC Error is Dropped

If the errored packet sits around in the crosspoint buffer long enough, we have time to cancel it before it goes out. In this example, we show how that would work. Consider the same 6-ford packet, but this time the XB was not able to send it out because of contention for the output port.

It is important to invalidate errored packets that are consuming crosspoint buffer entries. Before long, replay will provide the good version of the packet and try to put it in the same crosspoint buffer entry. If the entry is still filled by a bad packet, we would have to keep replaying until the junk packet wins arbitration and gets sent out.

Cycle	Description
S0	FORD1 arrives from Fabric Link Receiver.
S1	IBX checks for errors, modifies header, sends to XBX. XBX stores FORD1. FORD2 arrives from FLR, and input block continues to compute CRC.
S2	XBX chooses this packet in local arbitration and asserts ReqPst. OBX does global arb, and some other packet is selected. The XBX continues to request for this packet.
S3	FORD4 arrives in IBX. The XBX continues to request.
S4	FORD5 arrives in IBX. The XBX continues to request.
S5	FORD6 arrives from FLR. The XBX continues to request. The input block computes final CRC and it doesn't match the CRC field in FORD6. Now we know there was a CRC error! The IB changes the packet type to Poison as it sends to XB, asserts ibx_xbx_BadPacket_s1a, and increments a CRC error count register. The BadPacket signal causes the XB to clear the valid bit for the XB entry.
S6	Because the XB entry for the bad packet is no longer valid, the XB stops requesting in S6. There's still one last way that the packet will be sent out. If in S6, a GntPst arrives from the OBX, the XB would still have to send the poisoned packet. (Remember, grants always apply to the request that was made one cycle before.) Otherwise, the bad packet is dropped.

3.11.7 About the Bypass Paths

The canonical path through the fabric switch (above) has 6 cycles of latency, but our goal is 3 cycles of latency. When the switch is not busy and all required resources are available, packets can bypass the crosspoint buffer and go straight from the input block to the output block. But before we can accept a packet for bypass, we must check several things.

1. Availability of downstream buffers
2. Eligible packets in any XB contending for the same output port must go first, because they are clearly older
3. Packets arriving simultaneously in other IBXes destined for the same OBX (only one can bypass)
4. The output port may be busy
5. Arbitration in OBX may be disabled because a packet is going out already, or because we are in replay

Usually packets travel from IBX to XBX to OBX, but there are timing concerns about using this path in the minimum latency case. To solve this, in the start-of-packet cycle, the IBX forwards packet data and all necessary control signals directly to the OBX (in addition to the XBX) as it arrives. In this case, the IBX asserts ibx_obx_ReqBypS1_s1a to the OBX, and the OBX decides whether to allow the packet to bypass or not. Since the OBX does all arbitration between XBs and has complete information about which downstream buffers are free, the OBX will perform the checks for bypass eligibility as well.

Another common case is that a packet arrives while the OBX is busy, but it becomes free one or two cycles later. It's a shame to make these packets wait for the 6-cycle latency path when they could in theory go through in 4 or 5 cycles. To accomodate these packets that just missed the window of opportunity, we provide two other bypass options by delaying the data for one or two cycles in a pipeline in the OBX.

The four types of requests presented to the OBX are described below.

Request Signal	From/To	Description
ibx_obx_ReqBypS1_s1a	IBX to OBX	Request to send out the packet that is now being forwarded from IBX to OBX. If accepted, the latency is 3 cycles.
xbx_obx_ReqBypS2_s2a	XBX to OBX	Request to send out the packet that was forwarded from IBX to OBX one cycle ago. If accepted, the latency is 4 cycles. This type of request is only allowed if obx_xbx_PktCanBypass_s1a was asserted in the previous cycle.
xbx_obx_ReqBypS3_s2a	XBX to OBX	Request to send out the packet that was forwarded from IBX to OBX two cycles ago. If accepted, the latency is 5 cycles. This type of request is only allowed if obx_xbx_PktCanBypass_s1a was asserted in the previous cycle.
xbx_obx_ReqPst_s2a	XBX to OBX	Request to send out a packet from the XB packet store. Data will be sent from XBX to OBX one cycle later. If accepted, the latency is 6 cycles (assuming downstream buffers are available and packet wins all arbitration on first attempt).

The OBX considers all requests and chooses a winner. It drives the following signals to tell the XBX what is going on.

Grant Signal	From/To	Description
obx_xbx_GntPst_s3a	OBX to XBX	Indicates that the request made in the previous cycle was granted, and the XB should drive the packet data to the OB in the next cycle. The OB can assert GntPst in response to ReqPst. The XB invalidates the packet's XBE immediately to prepare for a new packet.
obx_xbx_GntByp_s3a	OBX to XBX	Indicates that the request made in the previous cycle was granted, and data was already present in the OB. The OB can assert GntByp in response to ReqBypS1, ReqBypS2, or ReqBypS3. The XB invalidates the packet's XBE immediately.
obx_xbx_PktCanBypass_s1a	OBX to XBX	PktCanBypass=1 tells the XB that it is allowed to use bypass requests ReqBypS2 and ReqBypS3 starting in the following cycle. If PktCanBypass=0, it can only assert ReqPst requests. It is valid all the time, generated based on the state of the output port.

In the following tables, the 3, 4, 5, and 6 cycle paths through the switch will be described.

3.11.8 3 Cycle Latency Path

This path shows how a packet would see minimum latency through the switch. If all the required resources are available, the packet can go straight from the input block to the output block with a total latency of 3 cycles (15 ns).

Cycle	**Description**
S0	SoP data arrives from Fabric Link Receiver on IB1 destined for OB2.
S1	IB1 checks for errors, modifies header, sends data to both XB12 and OB2. It asserts ib1_ob2_ReqBypS1_s1a. Because a downstream buffer is available and there is no contention, OB2 decides to allow bypass. The OB2 output mux selects the bypassed data from IB1, fills the LSN and CRC fields and flops the data. Now we can go straight to S5!
S5	OB2 drives data to Fabric Link Transmitter. All outputs are also stored in Replay Buffer. OB2 asserts ob2_xb12_GntByp_s2a, to tell XB12 that the incoming packet was selected for bypass. The XBX clears the valid bit on the XB entry into which the packet is (still) being written.

3.11.9 4 Cycle Latency Path

Cycle	**Description**
S0	SoP arrives from Fabric Link Receiver on IB1 destined for OB2.
S1	IBX checks for errors, modifies header, sends data to both XB12 and OB2. It asserts ib1_ob2_ReqBypS1_s1a. But OB2 is still sending out a packet, so the bypass request is rejected. OB2 places the data in its bypass delay pipeline.
S2	XB12 chooses the oldest eligible packet. If the incoming packet is selected and PktCanBypass was asserted in the previous cycle, XB12 asserts xb12_ob2_ReqBypS2_s2a. Let's say that the OB2 output port is no longer busy and the bypass packet is selected. OB2 reads from the S2 stage of its bypass pipeline, fills in LSN and CRC, and sends the packet out immediately. We can skip to S5!
S5	OBX drives data to Fabric Link Transmitter. All outputs are also stored in Replay Buffer. The OBX asserts ob2_xb12_GntByp_s3a to inform XB12 that its packet won. The XBX clears the valid bit on the XB entry into which the packet is (still) being written.

3.11.10 5 Cycle Latency Path

Cycle	Description
S0	SoP arrives from Fabric Link Receiver on IB1 destined for OB2.
S1	IBX checks for errors, modifies header, sends data to both XB12 and OB2. It asserts ib1_ob2_ReqBypS1_s1a. But OB2 is still sending out a packet, so the bypass request is rejected. OB2 places the data in its bypass delay pipeline.
S2	XB12 chooses the oldest eligible packet. If the incoming packet is selected and PktCanBypass was asserted in the previous cycle, XB12 asserts xb12_ob2_ReqBypS2_s2a. But the output port is still busy, so nobody wins.
S3	XB12 chooses the oldest eligible packet. If the incoming packet is selected and PktCanBypass was asserted in the previous cycle, XB12 asserts xb12_ob2_ReqBypS3_s2a. Let's say that the OBX output port is no longer busy and the bypass packet is selected. OB2 reads from the S3 stage of its bypass pipeline, fills in LSN and CRC, and sends the packet out immediately. We can skip to S5!
S5	OBX drives data to Fabric Link Transmitter. All outputs are also stored in Replay Buffer. The OBX asserts ob2_xb12_GntByp_s3a to inform XB12 that its packet won. The XBX clears the valid bit on the XB entry into which the packet is (still) being written.

3.11.11 6 Cycle Latency Path (No Bypass)

This is the canonical 6-cycle path through the fabric switch again. I include it to contrast it with the bypass paths. Here you can see how the bypass logic disables itself.

Cycle	Description
S0	SoP arrives from Fabric Link Receiver on IB1 destined for OB2.
S1	IBX checks for errors, modifies header, sends data to both XB12 and OB2. It asserts ib1_ob2_ReqBypS1_s1a. But OB2 is still sending out a packet, and there are 3 fords left to transfer so bypass is not going to help anybody. The output block always knows how many fords are remaining, and it uses that value to produce ob2_xbx_PktCanBypass_s1a. In every cycle, this signal tells crosspoint buffers whether they should use bypass requests or packet store requests in the next cycle. In this case, bypass is useless so PktCanBypass would be deasserted.
S2	XB12 chooses the oldest eligible packet. If any packet is eligible, XB12 asserts a request...but which kind of request? It considers using a bypass request, but it can't because PktCanBypass was off in the previous cycle. So, it raises xb12_ob2_ReqPst_s2a and starts to read the packet. Global arb selects XB12 as the winner.
S3	OB2 asserts ob2_xb12_GntPst_s3a to the winning XB12, which starts to read packet store address 1 of winning packet.
S4	XB12 delivers packet SoP to OB2. OB2 fills LSN and CRC fields and flops data.
S5	OB2 drives SoP data to Fabric Link Transmitter. All outputs are also stored in Replay Buffer.

3.11.12 End of Control Packet Arrives, Packets are Acknowledged

information propagates into the output block and crosspoint buffer. At first, imagine that the replay buffer contains 3 packets: LSN 6, 7, and 8. The replay write LSN is 9, so the next data packet will have LSN 9. The replay read LSN is 6. The last acknowledged LSN (AckLSN) is 5.

Cycle	Description
S0	The final byte of a control packet arrives at OB0. The data is flopped on the rising edge of S1.
S1	The CRC is checked, and the control packet is found to be good. Write the AckLSN, buffer busy masks, out of band data, etc. at rising edge of S2. The new AckLSN is 7, acknowledging correct receipt of LSNs 6 and 7.
S2	The new downstream busy mask is ORed with the pessimistic busy mask produced by the replay buffer, and driven from OB0 to its four crosspoint buffers, which flop the busy mask. In the replay buffer, the read LSN is compared compared with AckLSN. All LSNs up to and including AckLSN are acknowledged, and the busy mask bits for any buffers that the acknowledged packets consumed are cleared. The replay read LSN is set to 7.
S3	Crosspoint buffers may now make requests based on the busy bits from the new control packet. The pessimistic busy mask now shows that LSN6's and LSN7's buffers are available, so the busy mask sent to XBs may change.

3.11.13 End of Control Packet Arrives with ErrFlag=1, Causing Replay

At first, traffic is flowing normally from the crosspoint buffer and control packets are acknowledging packets without error. This corresponds to Replay State = NORMAL. Then a control packet arrives with ErrFlag=1 and starts the replay sequence.

Cycle	Description
S0	The final byte of a control packet arrives at OB0. The data is flopped on the rising edge of S1 in temporary registers.
S1	The CRC is checked, and the control packet is found to be good. Write the ErrFlag and AckLSN to registers that are visible to the output block.
S2	In the replay buffer, the read LSN is compared compared with AckLSN. All LSNs up to and including AckLSN are acknowledged as usual. But because ErrFlag is asserted, global arb is inhibited starting in S2. Any requests granted in previous cycles must be completed, but from S2 to the end of replay, no more grants will be issued. Let's assume a crosspoint buffer requested in S1 and won. In S2, the grant goes back to the crosspoint, and the packet is sent out the output mux during the next few cycles. Wait for the packet SoP to be sent before setting ErrAck. This guarantees that an idle packet with ErrAck does not sneak out before the last normal packet.
WaitToAck 1	The packet that won global arb in S1 has reached the output mux.
WaitToAck 2	The packet that won global arb in S1 is sent to the FLT. Now set ErrAck=1 and ReplayState=HANDSHAKE. The ErrAck will be carried downstream in Idle packets. Increment replay counter CSR.
Handshake	In HANDSHAKE state, global arb is still inhibited. Wait until a new control packet arrives with ErrFlag=0.
...	
Handshake S0	The final byte of a control packet arrives at OB0, containing the ErrFlag bit = 0. The data is flopped on the rising edge of S1 in temporary registers.
Handshake S1	The CRC is checked, and the control packet is found to be good. Write the ErrFlag and AckLSN to registers that are visible to the output block.
Handshake S2	Once ErrFlag=0, assuming there are packets in the replay buffer, the replay state is changed to REPLAY, and the replay loop counters are initialized to start at the first packet after the AckLSN. Begin to read the first FORD of the first packet to be replayed. If the replay bufer is empty, set ReplayState=NORMAL and skip to Done!
S3	Memory read cycle. Increment loop counters and start next read. FIXME: Add one more cycle of delay in HLM to match the Verilog.
S4	Data emerges from replay buffer. Do ECC correction, select replay data on the output mux, insert LSN (from the replay buffer address), compute CRC and flop it.
S5	First FORD of replayed data is sent to FLT. Unlike other data packets, don't record replay packets in the replay buffer!
Replay 1...N-1	Continue to increment loop counters, read replay buffer, and send packets back-to-back.
Replay N	Replay loop counter reaches the end of the last packet in the replay buffer. Set replay state=NORMAL.
Done!	Because replay state=NORMAL, global arb is enabled again. A packet store or bypass request could arb and win in this cycle.

3.12 FSW Registers and Definitions

3.12.1 Package Attributes

Package

chip_fsw_spec

3.12.2 Definitions

Defines

FSW

Constant	Mnemonic	Definition
32'd04	MINFORDS_PACKET	Minimum number of fords in a single packet.
32'd19	MAXFORDS_PACKET	Maximum number of fords in a single packet.
32'd16	LSN_MAX	How many LSN values are there? This determines the size of some memories.
32'd15	LSN_BITMASK	To make LSNs wrap around, AND it with this value.
32'd2	INITIAL_LSN	The LSN of the first data packet after an output block is reset. The output block initializes its LSN pointers to this value.
32'd1	INITIAL_LGSN	After reset, the Last Good Sequence Number register in an input block is set to this value. INITIAL_LGSN + 1 should equal INITIAL_LSN.
32'd64	VC_NP_RR_TABLE_SIZE	Size of the round robin table for every combination of VC and NextPort. There are 16 VCs and 4 NextPorts, so 64.
4'b1111	POISON_TYPE	The packet type field in the trailer that is recognized by the switch is the poison type. The fabric switch spec defines the poison value to be all ones.
32'd16	XB_NUM_ENTRIES	How many entries in a crosspoint buffer?
32'd4	NUM_PORTS	How many ports in a switch?
32'd16	NO_XBE_AVAILABLE	For functions that return a crosspoint buffer entry number, this value means that no crosspoint buffer was available.

3.12.3 Output Mux Select Choices

Enum

FswOutSel

Constant	Mnemonic	Definition
3'd0	IDLE	Send idle packets
3'd1	WINNING_XB	Send packets from the winning crosspoint buffer
3'd2	BYPASS_S1	Send packets from the S1 bypass pipeline
3'd3	BYPASS_S2	Send packets from the S2 bypass pipeline
3'd4	BYPASS_S3	Send packets from the S3 bypass pipeline
3'd5	REPLAY	Send packets from the replay buffer

3.12.4 Replay State Machine

Enum

FswReplayState

Constant	Mnemonic	Definition
2'd0	NORMAL	Normal operation. Global arb enabled, each packet written to the replay buffer as it is transmitted. Transition to HANDSHAKE if ErrFlag asserted.
2'd1	HANDSHAKE	Assert error acknowledge flag in idle packets. Global arb disabled. Acknowledge packets up to and including the LSN in control packets. Wait for ErrFlag to be deasserted and then transition into REPLAY.
2'd3	DELAY_REPLAY	One cycle delay between HANDSHAKE state and REPLAY state.
2'd2	REPLAY	Global arb disabled. Resend packets from the replay bufer starting at the acknowledged LSN + 1. When the SoP of the last replay packet is sent, return to NORMAL state.

3.12.5 Fabric Switch Control/Status Registers

This section defines all the CSRs for the fabric switch. All fabric switch registers are accessible through the SCB (Serial Configuration Bus). Verification code may also use "direct read" and "direct write" methods to access any register in zero simulation time.

The registers are organized into the following sections: registers that affect the operation of the FSW, performance and error counters, and control of interrupts.

3.12.5.1 Block Reset Register

This register allows each block of the FSW to be reset individually. Each block has an active-high signal which causes that block to change everything back to its initial state and ignore input traffic. The individual resets are provided so that if one link needs to be reset, only the blocks related to that link would need to be affected.

All blocks start out in reset after power-on, so the fabric will be idle until software deasserts reset to all the blocks it needs. Normally, software would enable every block by writing all zeroes. But the whole point of separate reset bits is to reset them separately so here are a few scenarios which would take advantage of this ability. If an output link is known to be bad, software can keep the OB and the four XBs that drive it in reset. Example: FLT1 is a bad link, so assert reset in bit 4 and bits 12+0*4+1=13, 12+1*4+1=17, 12+2*4+10=21, and 12+3*4+1=25. Or if software wants to reinitialize the DMA engine, it may want to disable all traffic to and from the DMA; in that case it would assert reset in DmaiReset and DmaoReset registers until the DMA is ready to receive packets. Finally, if software needs to reset an entire ICE9, it should also ask neighboring ICE9s to reset the part of their fabric switch that faces the device that was reset. In the three upstream ICE9's, the output block should be reset to clear the replay buffer and any lingering LSN state. (If you want to clear/drain old traffic, reset the four crosspoint buffers leading to the output block too.) In the three downstream ICE9's, the input block should be reset to clear the replay and LSN state.

NOTE: We will not verify 2ˆ27 combinations of reset signals. We will verify the poweron case, operation with a few blocks permanently disabled, and recovery after an upstream or downstream switch has been reset.

NOTE: When an input block is held in reset, it sends no control packets. This will cause the connected fabric link receiver to lose its heartbeat and go into retraining. Similarly, when an output block is held in reset, it sends no idle packets, and the fabric link transmitter will lose its heartbeat and go into retraining. To avoid this, the link can be reset as well.

Register
R_FswBlockReset
Attributes
-noregtest -kernel
Address
0xE_7D00_001C

Bit	Mnemonic	Access	Reset	Type	Definition
31:27					Reserved
26:12	XbReset	RW	0x7fff		One bit per crosspoint buffer. Bit 12+X*4+Y affects crosspoint buffer XY. (There is no crosspoint buffer XB33.) While XbReset is asserted, invalidate all crosspoint entries.
11:9	DmaoReset	RW	0x7		One bit per DMA output block. Bits 11:9 affect DMAO2,1,0. While DmaoReset is asserted, reset any state in the Dmao block.
8:6	DmaiReset	RW	0x7		One bit per DMA input block. Bits 8:6 affect DMAI2,1,0. While DmaiReset is asserted, reset any state in the Dmai block.
5:3	ObReset	RW	0x7		One bit per output block. Bits 5:3 affect OB2,1,0. Reset output block to initial conditions. While ObReset is asserted, the output block invalidates all entries in the replay buffer, sets read and write LSN to 2, sets AckLSN to 1 (pretending it's received 1 from control packets), and immediately cancels any packet that is in the process of being sent. In reset the OB will not update counters or error flags, and will not send Idle packets.
2:0	IbReset	RW	0x7		One bit per input block. Bits 2:0 affect IB2,1,0. Reset input block to initial conditions. While IbReset is asserted, the input block will set its LGSN to 1 and clear any error state. It will not update counters or error flags and will not send control packets.

3.12.5.2 Block Enable Register

Register
R_FswBlockEnable
Attributes
-noregtest -kernel
Address
0xE_7D00_005C

Bit	Mnemonic	Access	Reset	Type	Definition
31:27					Reserved
26:12	XbEnable	RW	0		One bit per crosspoint buffer. Bit 12+X*4+Y affects crosspoint buffer XY. (There is no crosspoint buffer XB33.) While XbEnable is low, ignore new packets and generate no requests.
11:9	DmaoEnable	RW	0		One bit per DMA output block. Bits 11:9 affect DMAO2,1,0. While DmaoEnable is low, ignore requests from crosspoint buffers so that no traffic is sent to DMA.
8:6	DmaiEnable	RW	0		One bit per DMA input block. Bits 8:6 affect DMAI2,1,0. While DmaiEnable is low, drop any incoming packets by disabling PortSel signals to XB and OB.
5:3	ObEnable	RW	0		One bit per output block. Bits 5:3 affect OB2,1,0. While ObEnable is low, the output block ignores all requests to send new data packets. It will continue to send Idle packets so that the out-of-band channel works. Any data transmission or replay that is in progess when Enable goes low will continue until it completes.
2:0	IbEnable	RW	0		One bit per input block. Bits 2:0 affect IB2,1,0. While IbEnable is low, the input block will drop any incoming packets by disabling PortSel signals to XB and OB. It will continue to send control packets so that the out-of-band channel works.

Bug2014: The FSW contains a logic bug which affects the behavior of IbEnable and DmaiEnable. When IbEnable[N] is low, the input block is supposed to block any packets from entering the crosspoint buffers, and it does that correctly. But, when an errored packet is detected in input block N and IbEnable[N] is low, input block N may incorrectly ask the connected crosspoint buffers to erase the packet that was most recently sent there, via the ibx_xbx_BadPacket signal. If there is a packet in the crosspoint buffer, it will be cancelled if it hasn't been selected to go out the OBX yet. The result is that a packet that was sent from IBX to XBX while IbEnable[N] was high MIGHT be erased from the crosspoint buffer, if it's still there when IbEnable[N] is set to low. The same goes for Dmai. The simplest and most likely software workaround is to always set IbEnable=7 and DmaiEnable=7 and never touch them. Or, before changing IbEnable or DmaiEnable bits from 1 to 0, ensure that all traffic has flowed out of the connected crosspoint buffers by watching the BusyMask values in captured control packets. But usually if you're going to set these bits to 0, you are expecting packets to be dropped anyway so it may not matter.

3.12.5.3 Input Block Mode Register

There are three mode registers. R_FswIbMode[X] describes the behavior of input block X.

Register

R_FswIbMode[2:0]

Attributes

-kernel

Address

0xE_7D00_0010 - 0xE_7D00_0018

Bit	Mnemonic	Access	Reset	Type	Definition
31:4					Reserved.
3:1	PktDecrementVc	RW	7		Packet decrement VC. When set, configures IBX to decrement VC field in header of a data packet. Bit assignment is for [Link-2,Link-1,Link-0]. The default value is SET.
0	PktCrcEna	RW	0		Packet CRC checking enable. When set, enables CRC checking on incoming data and idle packets.

3.12.5.4 Output Block Mode Register

There are three mode registers. R_FswObMode[X] describes the behavior of output block X.

Register

R_FswObMode[2:0]

Attributes

-kernel

Address

0xE_7D00_0000 - 0xE_7D00_0008

Bit	Mnemonic	Access	Reset	Type	Definition
31:1					Reserved.
0	CtrlCrcEna	RW	0		Control packet CRC checking enable. When set, enables CRC checking on incoming control packets.

3.12.5.5 PoolMask Register

There are three PoolMask registers. R_FswPoolMask[X] affects the behavior of the crosspoint buffers that drive output block X.

The PoolMask register specifies which buffers are dedicated and which are in the common pool. For example, the value 0xFFC0 indicates that crosspoint buffer entries 0-5 are dedicated to VCs 0-5, and entries 6-15 are pool. If traffic is sent on any VC which has no dedicated buffer, deadlock may result.

Register

R_FswPoolMask[2:0]

Attributes

-kernel

Address

0xE_7D00_0060 - 0xE_7D00_0068

Bit	Mnemonic	Access	Reset	Type	Definition
31:16					Reserved.
15:0	PoolMask	RW	0xFF00		Sets the PoolMask vector for an output port. The pool mask is 16 bits wide. If bit X is clear, then buffer slot X in each of the affected XBs is dedicated to traffic on VC X, so VC X can safely be used. If bit X is set, then buffer slot X in each the port's XBs is considered in the shared pool, and VC X must not be used. See the discussion of the XB, Section 3.10.7. The only useful settings consist of ones in the MSBs followed by zeroes in the LSBs, e.g. 0x8000, 0xF000, 0xFF00, 0xFFF0, 0xFFFE. The default of 0xFF00 is correct for 8 VCs. We expect that all PoolMask values in all ports in every node will be set to the same value.

3.12.5.6 Out-of-Band Upstream Register

There are three upstream registers. R_FswOobUp[X] is used to send and receive data to/from the upstream fabric switch via Fabric Link Receiver X. For a description of out-of-band communication, see section 3.5.1. Bits 9:0 are used to send data upstream. Bits 25:16 are used to receive data from the upstream switch.

Register

R_FswOobUp[2:0]

Attributes

-kernel

Address

0xE_7D00_0080 - 0xE_7D00_0088

Bit	Mnemonic	Access	Reset	Type	Definition
31:26					Reserved.
25	RecvEmpty	R	X		Empty flag from the upstream node
24	RecvTaken	R	X		Taken flag from the upstream node
23:16	RecvData	R	X		8 bits of data from the upstream node
15:10					Reserved.
9	SendEmpty	RW	1		Empty flag to be sent to the upstream node
8	SendTaken	RW	0		Taken flag to be sent to the upstream node
7:0	SendData	RW	0		8 bits of data to be sent upstream

3.12.5.7 Out-of-Band Downstream Register

There are three downstream registers. R_FswOobDown[X] is used to send and receive data to/from the upstream fabric switch via Fabric Link Transmitter X. For a description of out-of-band communication, see section 3.5.1. Bits 9:0 are used to send data downstream. Bits 25:16 are used to receive data from the downstream switch.

Register

R_FswOobDown[2:0]

Attributes

-kernel

Address

0xE_7D00_00A0 - 0xE_7D00_00A8

Bit	Mnemonic	Access	Reset	Type	Definition
31:26					Reserved.
25	RecvEmpty	R	X		Empty flag from the downstream node
24	RecvTaken	R	X		Taken flag from the downstream node
23:16	RecvData	R	X		8 bits of data from the downstream node
15:10					Reserved.
9	SendEmpty	RW	1		Empty flag to be sent to the downstream node
8	SendTaken	RW	0		Taken flag to be sent to the downstream node
7:0	SendData	RW	0		8 bits of data to be sent downstream

3.12.5.8 Output Block Status Registers

R_FswObxStatus[X] describes the state of the replay buffer in output block X.
Register
R_FswObxStatus

Attributes

-kernel

Address

0xE_7D00_00D0

Bit	Mnemonic	Access	Reset	Type	Definition
31:30					Reserved.
29	Ob2ReplayEmpty	R	X		OB2: Replay buffer is empty.
28	Ob2ReplayFull	R	X		OB2: Replay buffer is full.
27:24	Ob2AckedLsn	R	X		OB2: The last LSN that has been acknowledged by the downstream node.
23:20	Ob2NextLsn	R	X		OB2: LSN that the output block will use next, when building the next data packet.
19	Ob1ReplayEmpty	R	X		OB1: Replay buffer is empty.
18	Ob1ReplayFull	R	X		OB1: Replay buffer is full.
17:14	Ob1AckedLsn	R	X		OB1: The last LSN that has been acknowledged by the downstream node.
13:10	Ob1NextLsn	R	X		OB1: LSN that the output block will use next, when building the next data packet.
9	Ob0ReplayEmpty	R	X		OB0: Replay buffer is empty.
8	Ob0ReplayFull	R	X		OB0: Replay buffer is full.
7:4	Ob0AckedLsn	R	X		OB0: The last LSN that has been acknowledged by the downstream node.
3:0	Ob0NextLsn	R	X		OB0: LSN that the output block will use next, when building the next data packet.

3.12.5.9 Force Error Register

This register causes the circuit to intentionally produce errors that the fabric switch knows how to detect. This will help us to test the error detection logic and error handling software. Any kind of error that we can force appears in the register description below.

Here are the types of errors that we WILL NOT force in hardware, and the reason why we have chosen not to do it. I will go through the interrupt cause registers in order that they appear in the text. All OOB interrupts are triggered by software actions, so they don't need a force bit. We don't have special bits that force counters to wrap, because software can simply set the counter to MAX-1 and then force the event to occur once. VC decrement errors can be created by software by sending a packet with the DMA engine that has a VC=0 on a route that does a decrement. LengthErrMax is difficult for hardware to produce without screwing up the logic, so no force bit is provided. Single bit errors can be forced on all replay buffers using or ObFlipMemBits all crosspoint buffers using XbFlipMemBits; bit flipping in individual memories one at a time is not supported.

NOTE: There is a restriction on forced errors in output blocks. Only one of bits 8:0 (the output block force error bits) may be set at a time. For a given type of output block error, you can set WhichOb to all 1's to generate one error in each output block, but you cannot generate different kinds of errors at once. To guarantee predictable behavior, after writing ones into WhichIb or WhichOb, do not write the register again until the WhichIb and WhichOb bits go down.

Register

R_FswForceErr

Attributes

-kernel

Address

0xE_7D00_002C

Bit	Mnemonic	Access	Reset	Type	Definition
31:30	ObFlipMemBits	RW	0		These bits are XORed with bits 1 and 0 of every word of data being written to every replay buffer. This allows software to force single and double bit ECC errors in the replay buffers.
29:28	XbFlipMemBits	RW	0		These bits are XORed with bits 1 and 0 of every word of data being written to every crosspoint buffer. This allows software to generate single and double bit ECC errors in the crosspoint buffers.
27:24					Reserved
23:21	WhichIb	RWS	0		This field controls which input block will generate the error described in bits 16-20. Bit 21+X controls input block X. After the error is forced once in an input block, the corresponding WhichIb bit will be cleared.
20:17					Reserved
16	IbCorruptCtl	RW	0		Flip bit 0 of byte 15 of exactly one control packet.
15:13	WhichOb	RWS	0		This field controls which output block will generate the error described in bits 0-12. Bit 13+X controls output block X. After the error is forced once in an output block, the corresponding WhichOb bit will be cleared.
12:9					Reserved
8	ObCorruptIdleCrc	RW	0		Flip bit 0 of the CRC field in exactly one idle packet. NOTE: only one of bits 8:0 may be set at a time.
7	ObCorruptPktCrc	RW	0		Flip bit 0 of the CRC field in exactly one data packet.
6	ObMissingLsn	RW	0		Flip bit 2 of the LSN field in exactly one output packet, after computing the CRC. (In other words, the packet will have bad CRC.)
5	ObBadNumFords	RW	0		Force the NumFords field to the value 3 in exactly one output packet, after computing the CRC.
4	ObBadXbeTargetErr	RW	0		Flip bit 0 of the XbeTarget field in exactly one output packet, after computing the CRC.
3	ObProtocolErr	RW	0		Leave SoP deasserted for exactly one output packet.
2					Reserved
1	ObMissingDatavalid	RW	0		Deassert datavalid during the second ford of exactly one packet.
0	ObLengthErrMin	RW	0		Force EoP during the second ford of exactly one packet. (It will also be on during at the normal time.)

3.12.5.10 Bypass Enable Register

This register allows software to enable/disable each type of bypass mode. This setting affects all XBs and OBs.

Register

R_FswBypassEnable

Attributes

-kernel

Address

0xE_7D00_003C

Bit	Mnemonic	Access	Reset	Type	Definition
31:6					Reserved
5	XbEnableEccCorr	RW	0		Perform error correction and detection in all crosspoint buffers. When a packet is read from a crosspoint buffer and sent to an output block, the ECC of each FORD is checked; this guards against bit errors introduced in the crosspoint buffer RAM. (Note: For implementation reasons, this ECC logic lives in the output block.)
4	ObEnableEccCorr	RW	0		Perform error correction and detection in all output blocks. Whenever a packet is replayed, the ECC of each FORD is checked as it is read from the replay buffer; this guards against bit errors introduced in the replay buffer RAM.
3	DmaiEnableEccCorr	RW	0		Perform error correction and detection in all DMA input blocks. When a packet enters the FSW from the DMA, the ECC of each FORD is checked; this guards against bit errors introduced by the memory system or in the DMA TX port register file.
2	EnableBypS3	RW	0		Enable 5-cycle bypass path
1	EnableBypS2	RW	0		Enable 4-cycle bypass path
0	EnableBypS1	RW	0		Enable 3-cycle bypass path

3.12.5.11 Input Block Data Packet CRC Error Counter

One per input block.

Register

R_FswDataCrcCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_0020 - 0xE_7D00_0028

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Data Packet CRC error counter. This counter counts number of data packets with CRC errors. When the counter wraps around, a bit in the interrupt register is set.

3.12.5.12 Input Block Idle Packet CRC Error Counter

One per input block.

Register

R_FswIdleCrcCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_0090 - 0xE_7D00_0098

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Idle Packet CRC error counter. This counter counts number of CRC errors on idle packets. When the counter wraps around, a bit in the interrupt register is set.

3.12.5.13 Input Block Good Packet Counter

One per input block.

Register

R_FswPktCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_0030 - 0xE_7D00_0038

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Packet counter. This counter counts number of good (error-free) data packets received. When the counter wraps around, a bit in the interrupt register is set.

3.12.5.14 Input Block Poison Counter

One per input block.

Register

R_FswPktPoisonCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_0040 - 0xE_7D00_0048

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Packet poison counter. This counter counts number of data packets which were poisoned or dropped by IBX (not packets which had the poison type as they entered). When the counter wraps around, a bit in the interrupt register is set.

3.12.5.15 Output Block Control Packet Error Counter

There are three counters in the three output blocks. R_FswObCrcErrCounter[X] counts erroneous control packets in output block X.

Register

R_FswObCtlErrCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_0050 - 0xE_7D00_0058

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Output block control packet error counter. This counter counts the number of times the output block has detected a control packet with an error (CRC or loss of DataValid). The error counter increments on last byte of the control packet, so packets that are too short will not affect the count. When the counter wraps around, a bit in the interrupt register is set.

3.12.5.16 Output Block Replay Counter

There are three replay counters. R_FswObReplayCounter[X] counts replay events in output block X.

Register

R_FswObReplayCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_0070 - 0xE_7D00_0078

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Downstream replay counter. This counter counts the number of times the output block has gone into replay at the request of the downstream node. When the counter wraps around, a bit in the interrupt register is set.

3.12.5.17 DMA Input Block Packet Counter

One per DMAI block. R_FswDmaiPktCounter[X] counts packets sent from DMA input block X to the FSW.

Register

R_FswDmaiPktCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_00B0 - 0xE_7D00_00B8

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Packet counter. This counter counts number of packets received from the DMA. When the counter wraps around, a bit in the interrupt register is set.

3.12.5.18 DMA Output Block Packet Counter

One per DMAO block. R_FswDmaoPktCounter[X] counts packets sent from FSW to the DMA output block X.

Register

R_FswDmaoPktCounter[2:0]

Attributes

-kernel

Address

0xE_7D00_00C0 - 0xE_7D00_00C8

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	0		Packet counter. This counter counts number of packets sent to the DMA. When the counter wraps around, a bit in the interrupt register is set.

3.12.5.19 Upstream Control Packet Capture Registers

These registers allow software to view the control packets sent upstream. R_FswUpCtlCaptureX[Y] captures word X of the control packets sent by input block Y. Capture only occurs when software writes the CaptureEna bit in R_FswUpCtlWord3[Y].

Register

R_FswUpCtlWord0[2:0]

Attributes

-kernel

Address

0xE_7D00_01C0 - 0xE_7D00_01C8

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Word	R	x		Bytes 3-0 of the latest control packet. Byte 0 is in the least significant bits.

Register

R_FswUpCtlWord1[2:0]

Attributes

-kernel

Address

0xE_7D00_01D0 - 0xE_7D00_01D8

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Word	R	x		Bytes 7-4 of the latest control packet. Byte 4 is in the least significant bits.

Register

R_FswUpCtlWord2[2:0]

Attributes

-kernel

Address

0xE_7D00_01E0 - 0xE_7D00_01E8

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Word	R	x		Bytes 11-8 of the latest control packet. Byte 8 is in the least significant bits.

Register

R_FswUpCtlWord3[2:0]

Attributes

-kernel

Address

0xE_7D00_01F0 - 0xE_7D00_01F8

Bit	Mnemonic	Access	Reset	Type	Definition
24	CaptureEna	RWS	0		Whenever the CaptureEna bit transitions from 0 to 1, the next control packet will be captured into R_FswUpCtlWord0-3.
23:0	Word	R	x		Bytes 14-12 of the latest control packet. Byte 12 is in the least significant bits.

3.12.5.20 Interrupt Cause Registers 0, 1, 2

The interrupt cause register contains flags which are set when an event occurs, and cleared by software by writing a 1 to that bit. The FswIntCause[X] register reflects events that occur in input block X, output block X, and DMA input block X. While normally we would like to split these up so that all the bits come from the same block, they are grouped together here to reduce the number of registers that software has to read when an interrupt occurs.

Register

R_FswIntCause[2:0]

Attributes

-kernel

Address

0xE_7D00_0100-0xE_7D00_0108

Bit	Mnemonic	Access	Reset	Type	Definition
31					Reserved
30	IbRecvUpTaken	RW1C	0		The RecvTaken flag in the R_FswOobUp[X] register has toggled.
29	IbRecvUpEmpty	RW1C	0		The RecvEmpty flag in the R_FswOobUp[X] register has toggled.
28	IbPktPoisonCountWrap	RW1C	0		FswPktPoisonCounter[X] has wrapped around
27	IbPktCountWrap	RW1C	0		FswPktCounter[X] has wrapped around
26	IbIdleCrcCountWrap	RW1C	0		FswDataPktCounter[X] has wrapped around
25	IbDataCrcCountWrap	RW1C	0		FswIdlePktCrcCounter[X] has wrapped around
24	IbMissingLsn	RW1C	0		Missing LSN error. When set, indicates that at least once, the LSN of a data packet was not equal to the next number in the sequence.
23	IbBadNumFords	RW1C	0		Bad NumFords field error. When set, indicates that at least once, the NumFords field in the data packet header was not between FSW_MINFORDS_PACKET and FSW_MAXFORDS_PACKET.
22	IbVcDecrErr	RW1C	0		Virtual channel decrement error. When set, indicates that at least once, the virtual channel decremented below zero and the packet was redirected to the DMA.
21	IbBadXbeTargetErr	RW1C	0		Bad XbeTarget error. When set, indicates that at least once, the XbeTarget field indicated a crosspoint buffer that was already occupied.
20	IbProtocolErr	RW1C	0		Data packet protocol error. When set, indicates that at least once, SOP/EOP pair was not observed.
19					Reserved
18	IbMissingDatavalid	RW1C	0		Missing Datavalid during data packet. When set, indicates that DataValid signal has been observed missing during valid data packet.
17	IbLengthErrMin	RW1C	0		Min packet length error. When set, indicates that the EoP pulse arrived before the NumFords field specified.
16	IbLengthErrMax	RW1C	0		Max packet length error. When set, indicates that the EoP pulse did not arrive when the NumFords field specified. (Maybe it came later, or maybe not at all.)
15					Reserved.
14	ObReplayFull	RW1C	0		The replay buffer in OBX number X is full.
13	ObRecvDownEmpty	RW1C	0		The RecvEmpty flag in the R_FswOobDown[X] register has toggled.
12	ObRecvDownTaken	RW1C	0		The RecvTaken flag in the R_FswOobDown[X] register has toggled.
11	ObRepDoubleBitErr	RW1C	0		An uncorrectable error has occurred in the replay buffer in OB[X]. This means two or more bits were corrupted, and the ECC corrector could not fix it.
10	ObRepSingleBitErr	RW1C	0		A single bit error has occurred in the replay buffer in OB[X], and has been corrected.
9	ObReplayCountWrap	RW1C	0		FswObReplayCounter[X] has wrapped around
8	ObCtlErrCountWrap	RW1C	0		FswObCtlErrCounter[X] has wrapped around
7:5					Reserved.
4	DmaoPktCountWrap	RW1C	0		FswDmaoPktCounter[X] has wrapped around
3	DmaiPktCountWrap	RW1C	0		FswDmaiPktCounter[X] has wrapped around
2	DmaiBadNumFords	RW1C	0		DMA input block has detected a NumFords

3.12.5.21 Interrupt Cause Register 3 - For Crosspoint Buffer ECC Errors

Each of the 15 crosspoint buffers detects single bit ECC errors and double bit ECC errors. Each crosspoint buffer sends that information to the CSR module, which sets one bit in this register for each type. Bits 0 and 16 correspond to XB00, bits 1 and 17 correspond to XB01, etc.

Register

R_FswIntCauseXbEccErr

Attributes

-kernel

Address

0xE_7D00_010C

Bit	Mnemonic	Access	Reset	Type	Definition
31					Reserved.
30:16	DoubleBitErr	RW1C	0		There are 15 bits corresponding to 15 crosspoint buffers. If while reading XBmn, two or more bits are corrupted in a 64-bit word, bit number (16+4*m+n) is set. Such errors cannot be corrected.
15					Reserved.
14:0	SingleBitErr	RW1C	0		There are 15 bits corresponding to 15 crosspoint buffers. If a single bit error is found and corrected while reading XBmn, bit number (4*m+n) is set.

3.12.5.22 Interrupt Mask Registers

For each interrupt cause register, one interrupt mask register controls which conditions can cause the interrupt to be asserted. R_FswIntMask[2:0] enables interrupts for bits in R_FswIntCause[2:0]. R_FswIntMask[3] enables interrupts for bits in R_FswIntCauseXbEccErr. All bits are readable/writable, even though there are some bits for which there is not (yet) any cause bit.

Register

R_FswIntMask[3:0]

Attributes

-kernel

Address

0xE_7D00_0190-0xE_7D00_019C

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	IntMask	RW	0		If the corresponding interrupt cause bit is ever set, assert the interrupt.

3.12.5.23 Master Interrupt Register

This register summarizes the four interrupt cause registers, above. By reading R_FswIntMaster, software can decide which interrupt cause registers are worth reading.

Register

R_FswIntMaster

Attributes

-kernel

Address

0xE_7D00_004C

Bit	Mnemonic	Access	Reset	Type	Definition
31	Intr	R	x		Intr is the boolean OR of all other bits in this register. It is driven to the fsw_xxx_Int output port, through the CSW, and a few cycles later ends up in the Slow Interrupt Status Register in each L2 segment, R_CacxSlIntStat (section 7.18.9).
30:4					Reserved.
3:0	WhichIntCause	R	x		Each bit of this tells whether there are any unmasked interrupt cause bits in one of the four Interrupt Cause Registers. Specifically, WhichIntCause[X] is asserted when any bit in the expression (R_FswIntCause[X] & R_FswIntMask[X]) is set. For X=3, use (R_FswIntCauseXbEccErr & R_FswIntMask[3]).

3.12.5.24 Model Magic Register

This register only exists in the high level model. It allows verification code to perform special functions such as dumping out the state to a log file.

Register

R_ModelMagicFsw

Attributes

-noregtest

Address

0xE_7D00_0300

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	MagicOp	W	0		Write with value 1 to make SystemC dump state to log file.

3.13 Reset and Initialization

3.14 Internal Data Formats and States

The data formats for some internal buses are documented here in the spec to help the SystemC and Verilog models stay in sync with each other. The only people who would care about these formats are the SystemC and Verilog authors. Everyone else can safely ignore this section.

3.14.1 Encoding of Buses between FswCsr and FswIbx

3.14.1.1 CsrIbxStat - For csr_ibx_Stat_sa bus

Class

CsrIbxStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:42]	U0		Unused. Drive 0.
d0[41]	OobUpEmpty		Out of band Empty flag to be sent upstream
d0[40]	OobUpTaken		Out of band Taken flag to be sent upstream
d0[39:32]	OobUpChar		Out of band character to be sent upstream
d0[31:9]	U0b		Unused. Drive 0.
d0[8:7]	IbNum		Tells the IBX its block number: 0, 1, or 2. Purpose: These bits will get shifted into the MSB of route, as the route is shifted right by two places.
d0[6]	IbCorruptCtl		This bit is the IbCorruptCtl bit ANDed with the WhichIb bit in R_FswForceErr. If set, corrupt the next control packet and set ForceErrDone.
d0[5]	EnableIb		Enable the IB.
d0[4]	ResetIbLow		Reset the IB. This signal is active low.
d0[3:1]	PktDecVc		Packet Decrement VC.
d0[0]	PktCrcEna		Packet CRC checking enable.

3.14.1.2 IbxCsrStat - For csr_ibx_Stat_sa bus

Class

IbxCsrStat

Attributes

-allowunder

Bit	Mnemonic	Type	Definition
d1[63:0]	Oflr0_fsw_InDat_s0a		A copy of flr0_fsw_InDat_s0a, for OCLA and Performance Counter
d0[63:48]	U0		Unused. Drive 0.
d0[47]	Ofsw_flr0_NewCtlPkt_s3a		fsw_flr0_NewCtlPkt_s3a
d0[46]	Oflr0_fsw_SoP_s0a		flr0_fsw_SoP_s0a
d0[45]	Oflr0_fsw_EoP_s0a		flr0_fsw_EoP_s0a
d0[44]	Oflr0_fsw_Idle_s0a		flr0_fsw_Idle_s0a
d0[43]	Oflr0_fsw_DatVal_s0a		flr0_fsw_DatVal_s0a
d0[42]	Oflr0_fsw_MissionMode		flr0_fsw_MissionMode
d0[41]	OobUpEmpty		Out of band Empty flag received from upstream
d0[40]	OobUpTaken		Out of band Taken flag received from upstream
d0[39:32]	OobUpChar		Out of band character received from upstream
d0[31:24]	CtlDat		8 bytes of control packet data
d0[23]	NewCtlPkt		pulse during first cycle of control packet
d0[22]	ForceErrDone		in the cycle after CorruptCtl causes a control packet to be corrupted, ForceErrDone is asserted for one cycle to tell the CSR module to clear the WhichIb bit.
d0[21:13]	U0b		Unused. Drive 0.
d0[12]	IdleCrcErr		Received idle packet CRC error.
d0[11]	PktMissingLsn		Missing LSN error.
d0[10]	PktBadNumFords		Packet header had bad NumFords field.
d0[9]	PktVcDecrErr		VC decrement error.
d0[8]	PktBadXbeTargetErr		Packet header had bad XBE target field.
d0[7]	PktProtocolErr		Data packet protocol error.
d0[6]	PktLengthMismatch		Packet size does not match length field in header.
d0[5]	PktMissingDatavalid		Datavalid is missing during data packet.
d0[4]	PktLengthErrMin		Min packet length error.
d0[3]	PktForceEop		Max packet length error.
d0[2]	PktForcePoison		Force poison bit error.
d0[1]	PktRcvdGood		Received good (error-free) data packet.
d0[0]	PktCrcErr		Received data packet CRC error.

3.14.2 SCB Performance Events

The following events are trackable by SCB statistical event counting.

Enum

FswScbEvent

Attributes

-descfunc

Constant	Mnemonic	Definition
8'h00	CYCLES	Count every cycle. Drive 1 always.
8'h80	FLR0_SOP	SoP from receive link 0
8'h81	FLR0_IDLE	Idle from receive link 0
8'h82	FLR0_MISSIONMODE	MissionMode from receive link 0
8'h88	FLR1_SOP	SoP from receive link 1
8'h89	FLR1_IDLE	Idle from receive link 1
8'h8A	FLR1_MISSIONMODE	MissionMode from receive link 1
8'h90	FLR2_SOP	SoP from receive link 2
8'h91	FLR2_IDLE	Idle from receive link 2
8'h92	FLR2_MISSIONMODE	MissionMode from receive link 2

8'h98	FLT0_SOP	SoP to transmit link 0
8'h99	FLT0_IDLE	Idle to transmit link 0
8'h9A	FLT0_MISSIONMODE	MissionMode from transmit link 0
8'h9B	OB0_BYP_S1	Bypass S1 granted from OB0
8'h9C	OB0_BYP_S2	Bypass S2 granted from OB0
8'h9D	OB0_BYP_S3	Bypass S3 granted from OB0
8'hA0	FLT1_SOP	SoP to transmit link 1
8'hA1	FLT1_IDLE	Idle to transmit link 1
8'hA2	FLT1_MISSIONMODE	MissionMode from transmit link 1
8'hA3	OB1_BYP_S1	Bypass S1 granted from OB1
8'hA4	OB1_BYP_S2	Bypass S2 granted from OB1
8'hA5	OB1_BYP_S3	Bypass S3 granted from OB1
8'hA8	FLT2_SOP	SoP to transmit link 2
8'hA9	FLT2_IDLE	Idle to transmit link 2
8'hAA	FLT2_MISSIONMODE	MissionMode from transmit link 2
8'hAB	OB2_BYP_S1	Bypass S1 granted from OB2
8'hAC	OB2_BYP_S2	Bypass S2 granted from OB2
8'hAD	OB2_BYP_S3	Bypass S3 granted from OB2
8'hB0	DMA_FSW_SOP0	SoP from DMA port TX0
8'hB1	DMA_FSW_DATVAL0	DatVal from DMA port TX0
8'hB2	FSW_DMA_BUFAVAIL0	BufAvail from DMA port TX0
8'hB8	DMA_FSW_SOP1	SoP from DMA port TX1
8'hB9	DMA_FSW_DATVAL1	DatVal from DMA port TX1
8'hBA	FSW_DMA_BUFAVAIL1	BufAvail from DMA port TX1
8'hC0	DMA_FSW_SOP2	SoP from DMA port TX2
8'hC1	DMA_FSW_DATVAL2	DatVal from DMA port TX2
8'hC2	FSW_DMA_BUFAVAIL2	BufAvail from DMA port TX2
8'hC8	FSW_DMA_SOP0	SoP to DMA Port RX0
8'hC9	FSW_DMA_DATVAL0	DatVal to DMA Port RX0
8'hCA	DMA_FSW_RDY0	Rdy from DMA Port RX0
8'hD0	FSW_DMA_SOP1	SoP to DMA Port RX1
8'hD1	FSW_DMA_DATVAL1	DatVal to DMA Port RX1
8'hD2	DMA_FSW_RDY1	Rdy from DMA Port RX1
8'hD8	FSW_DMA_SOP2	SoP to DMA Port RX2
8'hD9	FSW_DMA_DATVAL2	DatVal to DMA Port RX2
8'hDA	DMA_FSW_RDY2	Rdy from DMA Port RX2
8'hFF		Reserved.

3.14.3 Encoding of Buses between FswCsr and FswDmai

3.14.3.1 CsrDmaiStat - For csr_dmai_Stat_sa bus

Class
CsrDmaiStat
Attributes
-allowunder

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:4]	U0		Unused. Drive 0.
d0[3]	EnableEccCorr		Enable single bit error correction and double bit error detection as data is read from the DMA engine
d0[2]	ResetDmaiLow		Reset the DMAI. This signal is active low.
d0[1]	EnableDmai		Enable the DMAI.
d0[0]	U0b		Unused. Drive 0.

3.14.3.2 DmaiCsrStat - For dmai_csr_Stat_sa bus

Class

DmaiCsrStat

Attributes

-allowunder

Bit	Mnemonic	Type	Definition
d1[63:0]	Odma_fsw_InDat_s0a		dma_fsw_InDat_s0a
d0[63:8]	U0		Unused. Drive 0.
d0[7]	Odma_fsw_DatVal_s0a		A copy of dma_fsw_DatVal_s0a, for OCLA and performance counter
d0[6]	Odma_fsw_SoP_s0a		dma_fsw_SoP_s0a
d0[5]	Odma_fsw_EoP_s0a		dma_fsw_EoP_s0a
d0[4]	Ofsw_dma_BufAvail_s3a		fsw_dma_BufAvail_s3a
d0[3]	BadNumFords		Packet has arrived with NumFords field out of range.
d0[2]	IncrPktCount		Increment DMA input block packet counter
d0[1]	DoubleBitErr		ECC corrector in DMA input block has detected a double bit ECC error.
d0[0]	SingleBitErr		ECC corrector in DMA input block has detected a single bit ECC error.

3.14.4 Encoding of Buses between FswCsr and FswObx

3.14.4.1 CsrObxStat - For csr_obx_Stat_sa bus

Class

CsrObxStat

Bit	Mnemonic	Type	Definition
d1[63:9]	U1		Unused. Drive 0.
d1[8]	ObCorruptIdleCrc		copy of ObCorruptIdleCrc from R_FswForceErr
d1[7]	ObCorruptPktCrc		copy of ObCorruptPktCrc from R_FswForceErr
d1[6]	ObMissingLsn		copy of ObMissingLsn from R_FswForceErr
d1[5]	ObBadNumFords		copy of ObBadNumFords from R_FswForceErr
d1[4]	ObBadXbeTargetErr		copy of ObBadXbeTargetErr from R_FswForceErr
d1[3]	ObProtocolErr		copy of ObProtocolErr from R_FswForceErr
d1[2]	U1b		Unused. Drive 0.
d1[1]	ObMissingDatavalid		copy of ObMissingDatavalid from R_FswForceErr
d1[0]	ObLengthErrMin		copy of ObLengthErrMin from R_FswForceErr
d0[63:57]	U0		Unused. Drive 0.
d0[56:53]	EnableBypS3		Enable 5-cycle bypass path. This is a 4-bit vector. Bit 53+x enables bypass from IBx for x=0,1,2, or bypass from the connected DMAI for x=3.
d0[52:49]	EnableBypS2		Enable 4-cycle bypass path. This is a 4-bit vector. Bit 49+x enables bypass from IBx for x=0,1,2, or bypass from the connected DMAI for x=3.
d0[48:45]	EnableBypS1		Enable 3-cycle bypass path. This is a 4-bit vector. Bit 45+x enables bypass from IBx for x=0,1,2, or bypass from the connected DMAI for x=3.
d0[44]	OobWrite		Ask OB to force a gap between data packets so that an Idle packet will be sent carrying the new Oob values. It stays on until OobWriteAck is sent by the OB. This ensures that the Oob channel is never completely starved.
d0[43]	EnableEccCorrXbData		Enable single bit error correction and double bit error detection on data as it is read from the crosspoint buffer
d0[42]	EnableEccCorrReplay		Enable single bit error correction and double bit error detection as data is read from the replay buffer
d0[41]	OobDownEmpty		Out of band Empty flag to be sent downstream
d0[40]	OobDownTaken		Out of band Taken flag to be sent downstream
d0[39:32]	OobDownChar		Out of band character to be sent downstream
d0[31:24]	U0b		Unused. Drive 0.
d0[23]	EnableOb		Enable the OB.
d0[22:21]	U0d		Unused. Drive 0.
d0[20]	U0c		Unused. Drive 0.
d0[19]	CtrlCrcEna		Enable CRC checking on control packets
d0[18:17]	DriveBadBits		Invert bits 1 and 0 of data written to replay buffer, to force ECC errors
d0[16]	ResetObLow		Reset the OB. This signal is active low.
d0[15:0]	PoolMask		Pool Mask.

3.14.4.2 ObxCsrStat - For obx_csr_Stat_sa bus

Class

ObxCsrStat

Attributes

-allowunder

Bit	Mnemonic	Type	Definition
d1[63:0]	Ofsw_flt0_OutDat_s2a		A copy of fsw_flt0_OutDat_s2a, for OCLA and performance counter
d0[63:59]	U0		Unused. Drive 0.
d0[58:56]	BypassPerfCount		Bit 56 is high when bypass S1 is granted. Bit 57 is high when bypass S2 is granted. Bit 58 is high when bypass S3 is granted.
d0[55:48]	Oflt0_fsw_CtlDat_s0a		flt0_fsw_CtlDat_s0a
d0[47]	Oflt0_fsw_NewCtlPkt_s0a		flt0_fsw_NewCtlPkt_s0a
d0[46]	Ofsw_flt0_SoP_s2a		fsw_flt0_SoP_s2a
d0[45]	Ofsw_flt0_EoP_s2a		fsw_flt0_EoP_s2a
d0[44]	Ofsw_flt0_Idle_s2a		fsw_flt0_Idle_s2a
d0[43]	Oflt0_fsw_DatVal_s0a		flt0_fsw_DatVal_s0a
d0[42]	Oflt0_fsw_MissionMode		flt0_fsw_MissionMode
d0[41]	OobDownEmpty		Out of band Empty flag from downstream
d0[40]	OobDownTaken		Out of band Taken flag from downstream
d0[39:32]	OobDownChar		Out of band character from downstream
d0[31:24]	U0b		Unused. Drive 0.
d0[23:20]	AckedLsn		The last LSN that has been acknowledged by the downstream node.
d0[19:16]	NextLsn		LSN that the output block will use next, when building the next data packet.
d0[15:12]	XbDoubleBitErr		In the ObxCsrStat bus going to output block N, bit 12+M is set if a double bit error is detected in data coming from crosspoint buffer MN.
d0[11:8]	XbSingleBitErr		In the ObxCsrStat bus going to output block N, bit 8+M is set if a single bit error is detected in data coming from crosspoint buffer MN.
d0[7]	ReplayEmpty		Replay buffer is empty.
d0[6]	ReplayFull		Replay buffer is full.
d0[5]	ForceErrDone		In the cycle after one of the FswForceErr bits that affect the output block causes a data packet to be corrupted, ForceErrDone is asserted for one cycle to tell the CSR module to clear the WhichOb bit.
d0[4]	OobWriteAck		Acknowledges the OobWrite signal in CsrObxStat. Asserted for one cycle when the OobWrite takes effect.
d0[3]	IncrCtlErrCount		Error in a control packet. The OB asserts this signal for one cycle when a control packet error is detected. If DataValid is missing, assert once in the following cycle. If a CRC mismatch is detected, assert once in the following cycle. Even if multiple errors are detected, only assert one time per control packet.
d0[2]	IncrReplayCount		OB asserts this signal to increment its ObReplayCounter. It is asserted during the cycle in which m_FltErrFlag_s2a=1 and m_FltErrFlag_s3a=0.
d0[1]	DoubleBitErr		The replay buffer has detected a double bit ECC error.
d0[0]	SingleBitErr		The replay buffer has detected a single bit ECC error.

3.14.5 Encoding of Buses between FswCsr and FswDmao

3.14.5.1 CsrDmaoStat - For csr_dmao_Stat_sa bus

Class

CsrDmaoStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:10]	U0		Unused. Drive 0.
d0[9:6]	EnableBypS1		Enable 3-cycle bypass path. This is a 4-bit vector. Bit 45+x enables bypass from IBx for x=0,1,2, or bypass from the connected DMAI for x=3.
d0[5]	EnableEccCorrXbData		Enable single bit error correction and double bit error detection on data as it is read from the crosspoint buffer
d0[4]	EnableDmao		Enable the DMAO.
d0[3]	EnableBypS3		Enable 5-cycle bypass path.
d0[2]	EnableBypS2		Enable 4-cycle bypass path.
d0[1]	U0c		Unused. Drive 0.
d0[0]	ResetDmaoLow		Reset the DMAO. This signal is active low.

3.14.5.2 DmaoCsrStat - For dmao_csr_Stat_sa bus

Class
DmaoCsrStat
Attributes
-allowunder

Bit	Mnemonic	Type	Definition
d1[63:0]	Ofsw_dma_OutDat_s2a		A copy of fsw_dma_OutDat_s2a, for OCLA and performance counter
d0[63:7]	U0		Unused. Drive 0.
d0[6]	Ofsw_dma_DatVal_s2a		fsw_dma_DatVal_s2a
d0[5]	Ofsw_dma_SoP_s2a		fsw_dma_SoP_s2a
d0[4]	Ofsw_dma_EoP_s2a		fsw_dma_EoP_s2a
d0[3]	Odma_fsw_Rdy_s1a		dma_fsw_Rdy_s1a
d0[2]	XbDoubleBitErr		Double bit error is detected in data coming from the attached crosspoint buffer
d0[1]	XbSingleBitErr		Single bit error is detected in data coming from the attached crosspoint buffer
d0[0]	IncrPktCount		Increment DMA output block packet counter

3.14.6 Encoding of Buses between FswCsr and FswXbx

3.14.6.1 CsrXbxStat - For csr_xbx_Stat_sa bus

Class
CsrXbxStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:18]	U0		Unused. Drive 0.
d0[17:16]	DriveBadBits		Invert bits 1 and 0 of data written to crosspoint buffer, to force ECC errors
d0[15:5]	U0b		Unused. Drive 0.
d0[4]	EnableXb		Enable the XB.
d0[3]	EnableBypS3		Enable 5-cycle bypass path.
d0[2]	EnableBypS2		Enable 4-cycle bypass path.
d0[1]	EnableBypS1		Enable 3-cycle bypass path.
d0[0]	ResetXbLow		Reset the XB. This signal is active low.

3.14.6.2 XbxCsrStat - For xbx_csr_Stat_sa bus

Class
XbxCsrStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:0]	U0		Unused. Drive 0.

3.14.7 Open issues

Chapter 4

DMA Engine Microcode

by Jud Leonard
[Last modified $Id: dmauc.lyx 43841 2007-08-28 19:09:39Z leonard $].

4.0.8 Package Attributes

Package

chip_dmauc_spec

Attributes

-dwaccessors

4.1 Introduction

The DMA Engine provides a high-bandwidth interface between the memory system and the fabric switch, relieving software of the low-level work of repetitively creating packets of memory data and injecting them into the fabric, or accepting packets from the fabric and distributing their payload to appropriate locations in memory.

This chapter describes the functions and interfaces of the DMA Engine which are implemented in microcode, and are therefore more or less subject to modification in future revisions of that microcode. The underlying hardware mechanisms are described in the DMAEngine spec.

The DMA Engine is designed to work closely with both privileged kernel-level device drivers and user-level library software to provide very low overhead transfers in a protected virtual memory environment. Low overhead requires that typical transfers can be initiated and completed without invoking kernel-mode or interrupt-level software at either sender or receiver, and that buffers need not be copied.

The DMA Engine provides two levels of communication between cooperating processes within the system:

- At the first level, user-mode software creates a small information packet on a command queue in its local memory. The DMA engine pulls the packet off the queue and injects it into the switch fabric with addressing to deliver it to the desired destination process and error checks to confirm error-free transmission. At the destination, the DMA engine stores the packet on a user-accessible event queue for processing by software.

- At the second level, rather than generating and processing packets directly, software sets up sufficient state in the DMA engines at both ends of a transmission to permit the hardware to generate packets at the transmitter and interpret them appropriately at the receiver. In this case, the DMA engines at both ends are responsible for managing memory addressing, including generation and verification of physical addresses, for fragmenting messages into packets, and for reassembly, relieving software of packet-level activity.

For more information about the MPI (Message Passing Interface) standard, visit http://www.mpi-forum.org.

4.2 Goals

We've tried to make the DMA engine to be as simple as practical, while achieving the following functions:

- It should be able to process the packets of outgoing and incoming messages without intervention by software.
- It should be able to keep a modest number of input and output messages in progress concurrently.
- It should dispose of incoming packets it cannot handle by presenting them to software with minimum overhead (< 100 ns).
- It should be able to pass outgoing packets from software to the fabric with minimum overhead (< 100 ns).
- It should be able to process several packets concurrently, overlapping multiple memory references.
- It should support local memory-to-memory transfers between address spaces on a single node. It should also provide a fast memory zeroing function.
- For large contiguous messages from one node to any other on an otherwise idle network, it should achieve 2 GB/sec.
- It must protect the integrity of user and kernel processes from unrelated naive, buggy, or malicious user processes running on the same system. It is not obliged to protect a user process from kernel-mode software on any node, nor from other processes with which it is communicating. It need not prevent covert channels or denial of service attacks.

4.3 Differences, Bugs, and Enhancements

4.3.1 Product and Chip Pass Differences

1. NEED IMPL: TWC9A records the address and syndrome of DRAM ECC errors, bug2157.
2. NEED IMPL: TWC9A fixes generation of bad ECC when ECC correction disabled and a 32-bit aligned packet is read, bug2396. R_SdmaEccMode bit 6 (CifCorrEna) enables ECC correction in CIF. This logic is only needed when the microengine does a BRD from a memory address with bit 2 set (32-bit realignment). When CifCorrEna is off and the microengine does a BRD from a memory address with bit 2 set, the ECC written into the DMA's internal memory (TX or COPY port packet buffer) is incorrectly forced to zero. Data with corrupted ECC may reach the FSW or main memory when the packet is sent. To workaround, leave CifCorrEna always set.
3. NEED IMPL: TWC9A fixes non-correction of ECC during 32-bit realignment operations, bug2403. When the CifCorrEna bit is on, and DMA is doing a read with 32-bit realignment, and there is a single bit error on the data from the CSW, the RTL does not correct the error. The RTL corrects the error inside the DmaCifDatacalg modules, but then incorrectly puts out the uncorrected data on cif_xxx_Data*[63:0] and into the next DmaCifDatacalg module. But the ECC bits on cif_xxx_data*[71:64] are the ECC consistent with the corrected data, so the resulting data appears to have just a single bit error. Workaround: None needed, as the error will be corrected at the destination of the DMA engine.
4. MIGHTFIX: TWC9A might double the size of the instruction memory, bug3390.
5. MIGHTFIX: TWC9A might fix a performance issue which requires a dead cycle between DMA packets headed into the FSW, bug597.
6. MIGHTFIX: TWC9A might fix DmaCif RDIO being corrupted by subsequent WTIO from the same core, bug1991. This can cause RDIOs to return corrupted data when followed immediately by a WTIO from the same CPU. I/O accesses from different CPUs are not affected, and SPCLs are not affected. When it happens, the WTIO overwrites the data before it can be sent back to the core, so the RDIO incorrectly returns the data from the WTIO. To avoid this, either issue a SYNC instruction between the RDIO and WTIO, or be sure to use the RDIO result before issuing the WTIO. All DMA addresses are affected (RA_DmaImem, RA_DmaDmem, RA_DmaAppIface0,1, etc.) except for those in the SCB range (RA_SDma*). The bug has only been observed when DMA is in the process of doing lots of block writes and the CSW is heavily loaded.
7. MIGHTFIX: Various possible microinstruction enhancements, bug3392, bug3393, bug3394, bug3395, bug3396.

4.3.2 Known Bugs and Possible Enhancements

4.4 Model

4.4.1 Terminology

4.4.1.1 DMA Context (formerly Process)

The DMA Engine is interacting at any time with the six processors on the same node, and each of those processors has activities running in user and kernel mode. For this discussion, we'll refer to each of those activities as a DMA context. The DMA engine keeps separate state and control information for each of 14 contexts, so as to minimize the extent to which those activities must use mutual exclusion to coordinate activities. There may be multiple Unix threads on one or more processors sharing access to a single DMA context. In this case, the software must manage concurrent access to the hardware.

The DMA engine uses a 4-bit *context number* (called *process index* for historical reasons) to uniquely identify the block of DMA engine state associated with a particular Linux activity. That state includes a 16-bit *process ID*, which can be used by software to uniquely identify the Linux activity which manages the DMA context. Whenever it receives a packet, the DMA engine uses the process index to select a block of process state, and compares the process ID in the packet to that in the selected state. A mismatch causes the packet to be treated as an unexpected packet, and a PID Mismatch event is stored on the event queue for DMA context number 0.

4.4.1.2 Thread

The execution model for the DMA Engine is a multithreaded state machine with a thread associated with each input or output port. Each thread is activated to process a packet as the necessary resources become available: transmit threads wait for an empty transmit buffer, receive threads wait for a full receive buffer. Each port has four packet buffers, which spend approximately equal times (˜100 ns) in memory references, processing by a thread, and moving into or out of the fabric. Queues support communication between transmit and receive contexts, on the one hand, and software on the other.

There are three threads associated with the three input ports, three more with the three output ports, two with the copy function (separately for memory read and write), one for queue management, and a specialized thread to serve I/O register accesses; total 10.

4.4.1.3 Handle

The DMA Engine is accessible to both kernel- and user-mode processes, and it accesses buffers in the virtual memory address space of whatever process it is serving. To keep this safe, applications describe accessible memory in terms of handles. A handle is an offset into a table of physical memory addresses (called the Buffer Descriptor Table, *BDT*, or the Route Descriptor Table, *RDT*) approved for use by each process. The tables are writable only by the kernel, and the BDT may contain contiguous groups of entries describing virtually-contiguous regions of memory. Handles are used to identify buffer regions, commands, and routes.

4.4.1.4 Packet

The data transport and switching machinery works on units of data called packets, which are individually addressed, carry separate error detection codes, and include up to 128 bytes of user payload. With overhead, packets may be as large as 152 bytes. Section 4.9 describes the various packet types supported.

Packets can be categorized into three major classes:

DMA Packets carry up to 128 bytes of message data between application-space buffers.

Command Packets carry instructions to be enqueued and processed by the receiving DMA engine; such commands are treated as if they had been issued by the receiving process at the destination node.

Interprocess Packets carry up to 128 bytes of data entirely determined by software, to be stored on the event queue of the receiving process.

4.4.1.5 Command

An instruction to the DMA engine, coming from a local processor or received encapsulated in a packet from a remote processor. Commands are stored on queues in memory while waiting to be performed by the DMA engine.

4.4.1.6 Segment

Messages may be very long – conceivably longer than the physical memory available to a single process. Therefore, we recognize that the message passing library software may want to break a single message up into a number of segments for independent transmission. The DMA engine hardware is optimized for the case that both source and destination buffers for each segment are available when that segment is transferred; that the transfer of an entire segment will be along a single path, with packets of the segment delivered in order; that most errors will be detected and corrected at the link level; and that uncorrected errors will be infrequent enough to justify retransmission of segments as a correction mechanism.

Segments serve an additional purpose as well: on lengthy transfers, we would like to distribute the traffic among disjoint routes from source to destination. The software on the originating node can fracture a message into multiple segments and transmit them along available routes to the destination in order to minimize overall message delay and hotspot congestion in the fabric. For very long messages, the software will enqueue later segments on the fly as earlier ones complete, to shift load to the fastest available path, and to avoid pinning too much memory at a time.

A segment may consist of a large number of packets, and we don't want to delay transfer of control information between nodes while waiting for completion of a segment, so segment transfers are treated as a background activity within the DMA engine; each output port generates packets for pending control transfers (foreground commands) in preference to segment transfers (background commands) on the same port.

4.4.1.7 Errors

While we recognize that packets will occasionally be corrupted and/or lost in the fabric, we have designed the low-level communication hardware to detect and retry corrupted packets, preserving their order, so we expect that failures at higher levels will be very rare events, and the system is designed to assume that all packets following a common path between any pair of nodes will be delivered uncorrupted in the order they were transmitted.

Note that the cut-through routing policy implies that a faulty packet may continue to propagate through the network, possibly even presented to a DMA engine for delivery at the incorrect destination. The switch is responsible for setting the type code of any corrupted packet to "poison", and the DMA engine is responsible for discarding any poisoned packet it receives.

The system is intended to make packet transfer sufficiently reliable that software can assume a transmitted packet will be delivered, and that foreground messages following a common path will be delivered in the order in which they were sent. Segment transfers can fail due to BDT faults at the source or destination nodes (indicating that a needed page has been swapped out); such faults are reported to software, which is expected to swap in the missing page and retry the transfer.

Software bugs can also prevent received packets from being processed correctly. In these cases, the hardware notes the errors in passing, and discards the packet.

4.4.1.8 Transmit

Within this chapter, *Transmit* (abbreviated *Tx*) is used to refer to the creation of packets and their injection into the switch fabric, typically starting in the application as MPI_SEND; so the transmit side of the engine is connected to the cache's Read Data bus; this can cause confusion, because of course the engine receives cache data to be transmitted through the fabric.

4.4.1.9 Receive

Similarly, *Receive* (abbreviated *Rx*) is used to refer to the whole process of acceptance, processing, and storage of packets coming from the fabric, starting in the application with MPI_RECV, and in the fabric with the arrival of a new packet; even though the engine must transmit memory addresses and data to the cache to store a packet.

4.4.1.10 Multicast

The DMA engine can be directed to produce several output packets directed to processes on various other nodes in response to a received packet, so that a group of processes can quickly inform members of the group about collective results. Multicast selectively targets processes so as to reach members of a group quickly without disruption to other groups.

4.4.1.11 Collective

The engine also implements a decrement and test function which allows another command to be triggered when a number of messages have been received; this permits the hardware to collect inputs from several sources and transmit when they have all been received.

4.4.1.12 Copy

The DMA engine is designed to support communication among application processes, whether they are on the same or different fabric nodes. To that end, the hardware supports local transfer of packets without use of the switch, but under the same protocol.

4.4.2 High-level Hardware View

The DMA Engine consists of a cluster of interacting state machines. The primary application interface consists of hardware-managed queues. One set of queues is used by the software to direct fabric activity, and another set is used by the engine to distribute incoming packets and completion events to the appropriate processes. The DMA engine is able to accept commands directly from any of the processors on the same chip, or indirectly from external processes through packets carried over the fabric.

The DMA engine has virtually no interest in the contents of packets, aside from the Route and the Packet type, which specifies the queue or buffer into which the contents are stored. Packet contents are fetched from and stored to contiguous blocks of memory.

All transfers are targeted to designated, pre-established destinations: either an event queue used by software, the DMA command queues used by DMA engine hardware, a reserved region of memory called the *heap*, or buffer specifically allocated for the transfer.

And just as a clarification: the DMA engine is not involved in processing packets which pass through the switch on their way between other nodes – it provides the path into and out of the switch fabric, but packets on their way from one node to another do not involve the DMA engine on intervening nodes along the path.

4.4.3 Canonical MPI Transfer Patterns

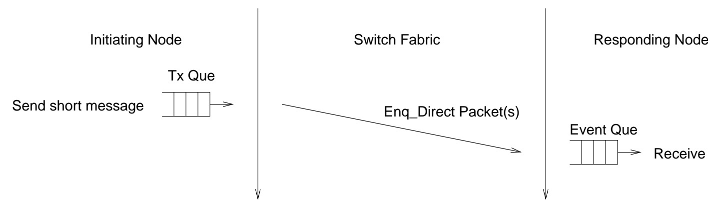
MPI provides three basic message transfer forms: Send/Recv, as specified in MPI-1, depends on the active participation of application software at both ends of a transfer. One process *Send*s a message to another process, which must perform a *Receive* to get it. The rules for matching sender and receiver essentially require the matching to occur at the receiver. The operation does not depend on the relative time order in which send and receive occur. The other forms, specified in MPI-2, are called Get and Put, and are described as single-ended because each message transfer is entirely specified by one process (the *Initiator*). The correspondent (*Responder*) declares a window in memory, and other members of the communicating group are permitted arbitrary access to that window.

4.4.3.1 Eager Transfer

For short messages, whether single- or double-ended, our goal is to complete the transfer with a minimum of overhead. Library software on the sending node queues a command to the local DMA engine for immediate transmission of a Enq_Direct packet which identifies the communicator, sender's rank, tag, and the data. Upon arrival at the remote destination, the remote DMA engine pushes the packet payload onto the event queue of the receiving process.

If the receiving process is waiting on a posted receive, the receiving process interprets the packet immediately. Otherwise, the packet is interpreted by a dedicated fabric processor, if there is one, or as a last resort, by a kernel-mode interrupt-level device driver. The receiving software is responsible for matching the communicator, rank, and tag of the packet with a posted receive, if there is one, and otherwise for storing the information to match against

later receives as they're posted. In eager transfers, the receiving software must copy the message contents to the destination buffer.



For intermediate-sized messages (too large for a single packet), software may choose to use Put_Im_Hp commands to copy from a buffer in the source application to the heap of the destination process, prior to notifying the receiver of message availability through an Enq_Direct.

4.4.3.2 Single-ended Messages

Once both ends of the communications link have set up buffer descriptors to describe the communications buffers, one-sided messages, *get* or *put*, may be used to move the data. If the sender initiates the transfer, a Put_Bf_Bf command is used, if the receiver initiates the transfer, a Send_Command containing a Put_Bf_Bf command is sent to the transmit-end DMA Engine.

Put_Bf_Bf waits in a transmit queue for access to the output port required by its route. When it reaches the head of the queue, it generates a sequence of DMA packets. When the DMA packets arrive at the receiver, the DMA Engine there places their contents in memory at the specified address. A special DMA_END packet terminates the transfer, at which point the receiving DMA Engine can execute a string of commands to signal software of completion, or store a fault event to signal failure.

4.4.3.3 Rendezvous Exchange

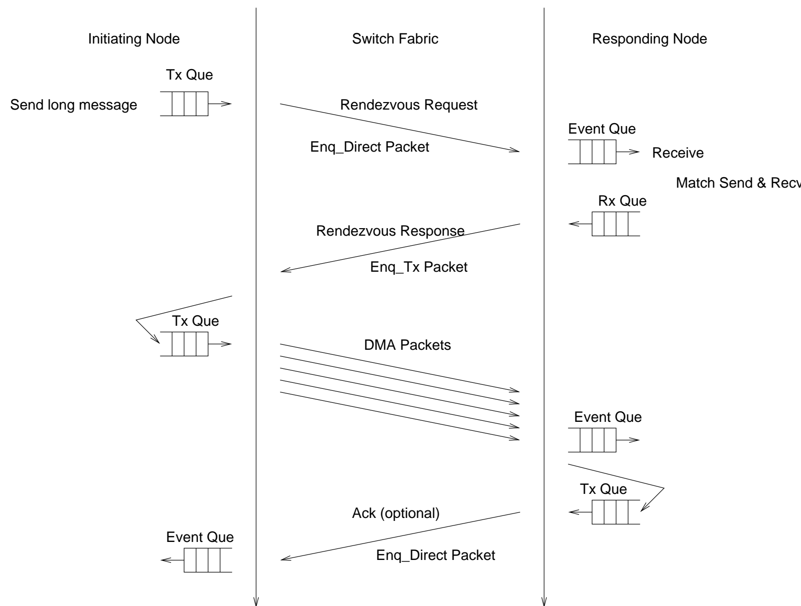
The sequence for Send/Recv transfer of a long message consists of an initial handshake called a *rendezvous*, in which the nodes agree that both are ready for the transfer to take place, with appropriate buffers available in memory and hardware resources for controlling the transfer.

The rendezvous exchange consists of a single Enq_Direct packet from the sender to the receiver in which the sender notifies the receiver of the existence of the message; its communicator, rank, and tag; and the BDT handles describing its buffer. When the receiver finds a matching receive, it performs the equivalent of a single-ended Get to transfer the message, except that the sender's DMA engine reports a completion event to the sender.

The rendezvous provides sufficient information for the sender and receiver to agree on the alignment of payload data within packets; the receiver acknowleges successful, error-free receipt of message segments, or requests retransmission of the segment in the event of a timeout or uncorrectable error.

For very long transmissions, the endpoints may agree to transfer several segments concurrently along disjoint paths, distributing the traffic around any hotspots.

The rendezvous exchange enables very efficient use of hardware, compared to a software-mediated (*eager*) transfer, but requires an additional trip to set up.



Rendezvous transfer described To transfer a long message using MPI_SEND/MPI_RECV, the sequence resembles the following:

- The sending application process calls MPI_SEND.
- The sending MPI library decides that the message length is great enough to justify rendezvous protocol (a compile-time parameter).
- The sending MPI library builds a Send_Event command which describes the communicator, sending rank, and tag of the message, along with a buffer handle and offset for the user's message buffer. This information is collectively called a rendezvous request. The library code pokes the DMA engine to tell it there's a command on the command queue.
- The DMA engine pops the Send_Event command from the process command queue, and translates its route handle to determine which output port should be used to reach the receiving node. If the foreground context for that port is available, the command is enabled for immediate output; otherwise it is copied to the port-specific transmit foreground queue for transmission as available.
- The Send_Event command results in delivery of an Enq_Direct packet to the receiving node, where it is matched to the target DMA context and stored on the event queue of that context.
- At some time either before, during, or after all the above, the receiving application process calls MPI_RECV.
- The receiving MPI library searches the lists of previously-unmatched Sends. If there is one whose communicator, rank, and tag match the parameters of the current receive, the match is made, and the receiver initiates a Get_Seg sequence, described below. If there is no match, the parameters of the current receive request are stored to be matched against future sends.
- The receiving MPI library processes the event queue. If it finds a send (either rendezvous or eager), the library searches the lists of posted receives to find a match.
- Once a match has been made, the library software at the receiver builds a Put_Bf_Bf command, which consists of two parts: the information needed by the receiver context to accept DMA packets for the transfer; and information needed by the sending node to build those DMA packets. Library software enqueues the Put_Bf_Bf command inside a Send_Cmd command.
- The receiver's DMA engine sends an Enq_Response packet to the sender, carrying the Put_Bf_Bf command to be executed to perform the transfer.
- When the Enq_Response arrives at the transmitter, it is enqueued to be performed when reaches the head of the background queue for the appropriate port.

- The sending node generates DMA packets as rapidly as the switch fabric can accept them, and the receiving node stores them in the destination buffer according to the receive context.
- Upon successful completion of the transfer, the receiver performs an optional command string.

4.5 Queues

The software interface to the DMA Engine consists of a page of control registers which are used by the kernel's device driver for configuration setup and diagnostic purposes, plus a set of control pages through which the library software requests activities by the engine, and through which the engine reports completion of requests and arrival of new messages. The hardware supports concurrent interaction with 14 DMA contexts, so there are 14 separate control pages as described in Table **??** below.

The control pages provide a multiport interface to a hardware queue manager which schedules the activities of the fabric input and output links. It accepts commands from fourteen DMA contexts and responses from three input links, distributing them into separate queues for each of the output links. It also provides a queue bypass function which avoids memory writes and reads in the (common) event that the target port is idle.

The memory area allocated for queues should be large enough to make queue overflow very unlikely, but the hardware will discard any received packet destined for a queue which doesn't have room for it. It is up to software to ensure that queues do not overflow; we expect that quotas will be used to ensure that there is space for every queue entry. Each process is allocated a quota which determines the maximum number of commands it may have in the port queues at any time; any commands in excess of that limit remain in the process command queue.

For simplicity of software (but not minimal memory use) all queue entries are 128 bytes, a multiple of the L2 cache block size, and are allocated aligned to cache blocks. This avoids issues of false sharing between entries. Software writes queue entries on the command queue by writing the entry in main memory. The hardware is informed of the update by a write to a special I/O register. Hardware then reads the command block to see which output port it needs. (See Figure 5.6)

The block is copied from the command queue, where it was written by software, to the port if idle, or to the selected port queue.

Port threads are responsible for pulling commands off the port queues as earlier commands complete. A specialized thread, called the queue manager, accepts commands as they are written by software, sorting them into the appropriate port queues or inserting them directly into available slots for use by transmit threads.

Each queue is described by a set of three values accessible to the kernel:

1. The memory region used for a queue is described by a buffer descriptor (see paragraph 4.7.4) with the physical address in bits 35:0, and the negative length of the region in bits 63:36.
2. The read pointer is the physical address of the next item to be removed from the queue (the head of the queue). If the queue is empty, the read pointer matches the write pointer.
3. The write pointer is the physical address at which the next item should be inserted in the queue (tail).

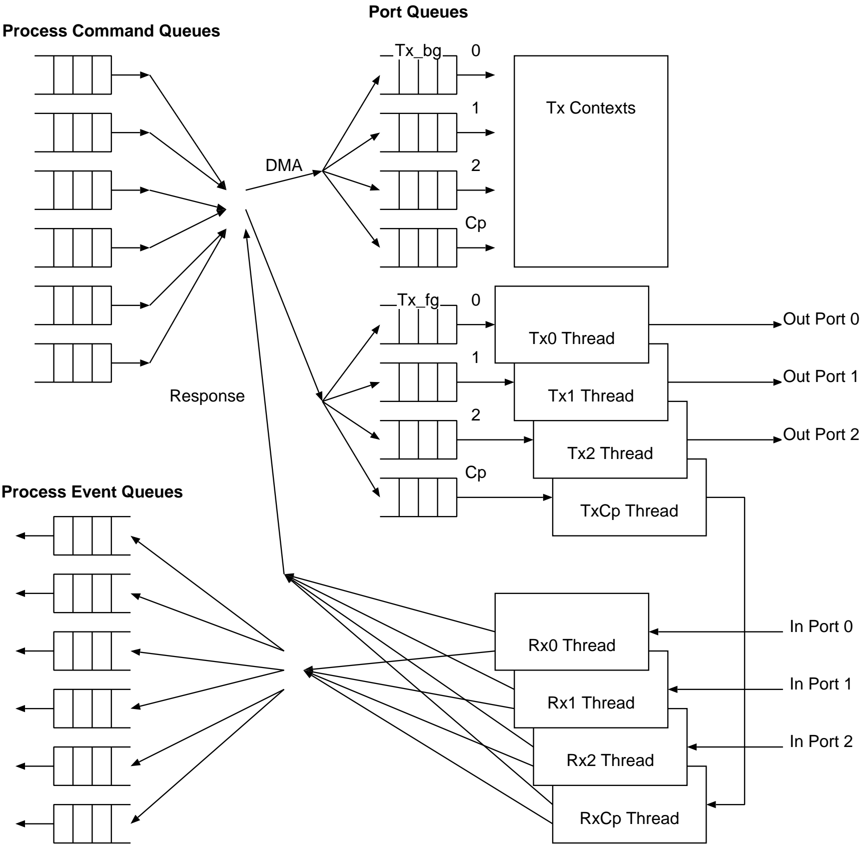
Both read and write pointers are incremented by 128 until the pointer reaches the end of the memory region, then it wraps back to the beginning of the region before reading or writing the next entry. The region descriptor length should be a multiple of 128.

4.5.1 Command and Port queues

The command queue is the mechanism by which applications software directs operation of the DMA Engine. To send a message, the software writes one or more commands, indicating the location of the data to be used (by buffer descriptor index, offset, and length), the destination (by route handle), and linkage to appropriate completion notice. Software notifies the DMA Engine of an addition to the command queue, using an I/O write to fastCmdHdr in DmaAppIface0 or cmdQWrSize in DmaAppIface1, and the DMA Engine either executes the command immediately or transfers the entry to the appropriate port queue. For single-packet message transmission, the command queue item typically contains the entire packet payload; microcode translates the route handle to obtain the routing information, assembles a packet, and appends a check code before injecting the packet into the fabric.

Software can directly add to the command queue on the local node. Those commands include the ability to enqueue commands at remote nodes as if they had been initiated by software on that node. This feature is used for single-ended operations and broadcast, among other purposes.

Figure 4.1: Command and Event Queues



Each process has one command queue which provides access to all the port queues. A transmit command may use the foreground queue for short control messages; a foreground command takes priority on the output port it needs, and gets sent as quickly as possible, but in order with respect to other foreground commands on the same port. The DMA engine thread for each output port services the transmit queues for that port on a foreground/background basis, servicing foreground transmits in preference to background.

Interface software must exercise care not to overrun pending commands on the command queue, and because commands for different ports may be serviced out of order, neither cmdQRdPtr nor cmdQRdOffset is a reliable indication of where the oldest pending command is. Software should use Supervise commands to determine how much of the command queue region is free.

4.5.1.1 Process quota

The port queues are shared among all processes on a node, so it is important to prevent bugs in one process from interfering with another; in particular, we must prevent overflow or saturation of the port queues by one process from damaging another. Therefore, each process is given a quota representing the maximum number of commands it may have in the port queues at any time, and the port queue regions must be sized to permit the full quota allocated to all processes in each of the port queues.

The DMA engine suspends processing of the command queue of any process which has reached its quota of commands in the port queues, and commands received from remote nodes for such a process are enqueued on the event queue rather than the port queue. Library software is expected to copy such deferred commands to the command queue, keeping them in order. The DMA engine maintains a count of the number of commands deferred in this way, and continues deferring remote commands to the event queue until all deferred commands have been enqueued to the port queues.

4.5.1.2 Command order

The DMA Engine provides a limited set of assurances about the order of command processing:

- Commands from a single process, sent out a single transmit port, will be sent in the order in which they are queued, except that background commands (Put_Bf_Bf) may be delayed with respect to newer foreground commands (any others).
- Foreground commands in a string invoked by Do_Cmd or a receive completion and directed to a single transmit port will be performed in order, but not necessarily ordered with respect to the command queue.
- Commands for multiple contexts or directed out different transmit ports are not ordered.

Combined with the assurances by the fabric of reliable, in-order delivery of packets following the same route and virtual channels, these conditions are sufficient for the software to ensure consistent ordering of messages where necessary.

4.5.2 Event queue

The event queue is the mechanism by which the DMA engine notifies software about completion of commands or errors which prevent completion, and also one of the mechanisms by which software on one node can communicate with another. Software can select whether the queueing of events raises an interrupt request (see paragraph 4.7.7). Typically, an entry on the event queue indicates that the transfer described by a transmit or receive context is complete, or that a remote process has sent a short message directly to this one.

4.5.2.1 Hardware-generated events

- Buffer descriptor invalid
- Unmatched Process ID
- Heap/BDT/RDT index out of bounds
- Diversion for port queue quota
- Segment completion at transmitter/at receiver

In general, fault events are delivered to the event queue which belongs to the local process which encountered the fault. When a Put_Bf_Bf command encounters a tx buffer descriptor fault, the transmitting node sends an Enq_Direct packet whose payload is stored at the receiver as a SegAbort event.

The first word of an event queue entry contains the event type in bits 11:8:

Enum

DmaEventType

Attributes

-allowlc -kernel

Constant	Mnemonic	Definition
4'd1	heapFault	A heap handle exceeds the heap size. The bad heap handle is stored in d1[31:0]
4'd2	rdtFault	A route handle exceeds the RDT size. The bad route handle is stored in d1[31:0]
4'd3	bdtFault	At the receiver, a buffer handle exceeds the BDT size, a buffer descriptor length is too short for the offset requested, or a buffer descriptor is marked read-only. The swBucket is stored in d1[63:0].
4'd4	cmdFault	An illegal command code was encountered. Either the command is undefined, or it was inappropriate to be issued as a fastCmdHdr. The command header is stored in d1[63:0].
4'd5	segAbort	Reported at the receiver when the transmitter aborted the segment. The swBucket is stored in d1[63:0].
4'd6	pidMismatch	A received packet contained the wrong process id for its selected process index. The packet header and trailer are stored in d1 and d2 on the process 0 event queue.
4'd7	queueFault	Software error setting up command or event queue pointers. This event is stored on the process 0 event queue. The process index of the failing process is stored in d1.
4'd8	deferredCmd	Process received more commands from remote nodes than allowed by the port quota; any excess are stored on the process event queue. This event queue entry contains, in d1 up to d14, the payload (a nested command) of an Enq_Response packet which could not be pushed onto the port queue.
4'd9	rxEndSeg	Successful end of segment at receiver. d1 contains swBucket.
4'd10	portFault	The txPort hint in a command header differs from the port specified by the route descriptor in the RDT. The command header is stored in d1.

Event queue entry Class

DmaEventQueue

Bit	Mnemonic	Type	Definition
d0[7:0]	eventLength		The "useful length" of the event queue entry, in bytes
d0[11:8]	eventType	DmaEventType	Event type code
d0[63:12]	reserved		Zeros
d1[63:0]	eventData		Information specific to the event type, as described above

Event queue entries are written 128 bytes apart, to keep the pointer management as simple as possible. The event length field indicates the number of bytes of the entry which were actually written by microcode.

4.5.3 Summary of DMA Engine Queues

To wrap up the section on queues, Table 4.1 is a list of all the types of queues that DMA engine interacts with. Below the table are some notes on the commands or events which are found in each queue. [from Bryce: When commands and events are more completely defined elsewhere, some of this should be removed.]

Table 4.1: DMA Engine Queues

Name	Contents	Writer	Reader	How many?
process command queue	commands	core	dma	14: one per Dma Context
process event queue	events	dma	core	14: one per Dma Context
transmit foreground port queue	commands	dma	dma	four: one per TX port, plus copy
transmit background port queue	commands	dma	dma	four: one per TX port, plus copy

Commands can contain:

- command type
- data to set up transmission
- raw packet data (can contain nested commands for remote DMA)

Events can contain:

- event type
- info about a transfer that completed or failed
- info about an unsolicited packet that arrived
- raw packet data

4.6 Modes of Operation

The DMA Engine hardware needs attention from a programmable processor at the beginning and end, and occasionally in the midst, of a message transmission. Under various circumstances, the processor selected to do the work might be the one running the application process, one dedicated as a fabric support processor, or an interrupt service routine in a designated processor. We distinguish these cases as *modes* because the literature refers to heater mode, communication processor mode, etc, to describe similar configurations, but unlike other cases in the literature, our system switches among the modes freely for optimal performance.

4.6.1 Synchronous mode

The conceptually simplest form of communication between MPI processes is syncronous mode, in which the sender creates and sends a message, waiting to proceed until it has been received, and the receiver declares an available buffer for the message, waiting until it has been filled.

In syncronous mode, the processors used by the communicating processes are essentially idle while the communication is going on, and are therefore the ideal candidates to perform any support and supervisory work required by the DMA hardware. In the current vision, that includes on the transmit side: maintenance of data structures, confirmation of error-free transmission, and timeout monitoring. On the receive side, it includes selection and scheduling of message segments; communicator, rank, and tag matching (*CRT match*); management of unexpected message buffers; maintenance of data structures, and timeout monitoring.

4.6.2 Asynchronous mode

Synchronous mode MPI communication allows no overlap between computation and communication, so MPI also provides asyncronous versions of both Send and Receive to permit the programmer to initiate one or several message transfers, conduct independent calculations, and then wait for completion of some or all of the transfers. Those portions of the transfer which take place when the application has finished its calculation and is waiting are treated as synchronous, in spite of having been initiated with the asynchronous calls; but for the remainder, we don't want to slow down the application by interrupting it to service the message.

Therefore the preferred mechanism for dealing with asyncronous message service is to designate one processor as the "fabric processor", and run it in a spin loop monitoring the input/event queues for all the others, and servicing traffic for each as it comes in.

4.6.3 Interrupt mode

Of course, there are times when there's nothing to do but compute, and lots of it. During those times, we would hate to have 1/6 of our compute capability tied up as a fabric processor, so we will return the fabric processor to the scheduling pool and handle any rare requirements for DMA Engine service as interrupt requests directed to a designated processor.

4.6.4 Fabric Processor

During those times that the system dedicates one processor on a node as the fabric processor, it will run a process which has mapped the heap, event queue, and buffer descriptor table of each application process into its own address space. The fabric processor and application processor interlock access to the event queue by means of shared variables in the heap to ensure that exactly one of them services every event.

4.6.5 Virtualized mode

It would be a desirable feature if the software were able to multiplex the limited hardware resources among a larger number of processes, so that descheduled processes (in the Unix sense) were still available to participate asynchronously in MPI communications. We have had some preliminary discussions about this possibility, but have not resolved all the protection issues involved. Two models have been discussed:

- Multiplexed applications are linked with a different library, which calls a daemon or kernel service to communicate, in a manner similar to MPI over TCP.
- Multiplexed applications timeshare a DMA Context for command and event queues, but external traffic is actually directed to a kernel-mode driver which demultiplexes to the appropriate address space. [How to handle remote commands?]

At the moment, this feature is mostly pipe-dream, but if we can devise a reasonable implementation, it would be desirable.

The simplest implementation of virtualization is provided by the currrent specification: to share the hardware resources, the operating system kernel stops all the processes of a job, waits for the job's current traffic to quiesce, and reassigns the hardware resources to the processes of a new job.

4.7 Communication state

Communicating processes may have a very large number of simultaneously-outstanding message requests; it is up to the MPI library or equivalent software to schedule message activity, and provide the DMA Engine with descriptive information about each active message.

In the descriptions which follow, unused or unspecified fields in commands and registers should be initialized as zero.

4.7.1 Transmit state

The DMA Engine maintains for each output port some transmit (Tx) state in a hardware structure which describes an outgoing segment during its transfer: a sequence of packets, the buffer from which they are read, and their destination, which typically consists of a route to a node and a receive context id on that node. (Table **??**) It is loaded by the transmit thread, which assembles the various components from the command, the buffer descriptor table, and the route descriptor table. When the transmit state has been loaded, the transmit thread is able to create packets and inject them into the fabric. When a complete segment has been transmitted, a new command is popped off the transmit queue.

When a transmit command is executed by the DMA engine, the Route Handle is used to lookup a route in the kernel-controlled route table, and the Buffer Handle is used to obtain the base address and length of a physically-contiguous region of the buffer. That region may not be as large as the message segment; if it runs out before the end of the segment, the DMA engine hardware increments the Buffer Handle to obtain a new BDT entry in which to continue the segment. The engine also clears the offset, so that subsequent packets will come from the beginning of the next region of buffer.

The DMA engine needs storage for 8 separate transmisions in hardware: foreground (bypass) and background (bulk) contexts for each output port plus the copy thread. The transmit queues of waiting commands are kept in memory queues associated with each port. The port-specific queues are written by the queue manager and read by the port threads as hardware space become available.

4.7.2 Receive state

Every DMA packet carries a 64 bit control word, which contains a buffer handle (2 bytes), a buffer offset (4 bytes) and a notifier (2 bytes). To work efficiently, the microcode implements a buffer descriptor cache with lookup faster than loading the BD from memory for each packet. This design makes it impossible to carry from one Rx buffer handle to the next in the middle of a segment. Software will arrange that DMA packets are full cache-line aligned at the receiver, and segments do not cross page boundaries at the receiver, so this won't be needed.

Segment transfers can fail because of BDT faults at transmitter or receiver. An attempt to access an invalid buffer descriptor or to write beyond the end of the buffer descriptor will be detected at the receiver. The receiver will set a bit in the heap selected by the notifier, and discard the packet. At the end of the segment, the transmit microcode will send a DMA_END packet, which causes the receiver to test the heap for an earlier error. If the transmit end faults due to a bad buffer descriptor, an ENQ_DIRECT packet with a Seg_Abort event will be sent to the receiver.

4.7.3 Notifiers

DMA commands include a 16-bit field, called the notifier, which is used by software to uniquely identify a segment transfer. In the event of a bdt failure at the receiver, the rxNotifier is used to remember which segment failed, and upon completion of the transfer, an entry is created on the local event queue, including the notifier of the failing DMA command and the bdt index responsible for the failure.

4.7.4 Buffer descriptor

Translates a process virtual address range to a contiguous physical address range. Used to describe message buffers, get/put windows, and queue rings. Contiguous groups of entries are used to describe contiguous regions of virtual address space which may be discontiguous in physical memory.

More particularly, each DMA Context has a register representing the starting physical address and length of the buffer descriptor table for that context (see Table **??**). The Buffer Descriptor Table (BDT) contains 8-byte entries, which contains the starting physical address of a buffer and the length of the buffer in bytes. A Buffer Handle, which appears in DMA command queue entries, is a 16-bit unsigned integer less than the BDT size; the hardware multiplies it by 8 and adds it to the bdtRegion pointer to identify a specific BDT entry.

A single BDT entry describes a region of memory which is contiguous in both virtual and physical address spaces; it is not necessarily restricted to a single page, though of course such a restriction is sufficient to ensure contiguity.

Each BDT entry is valid if its length field is negative. The DMA engine will abort transmission of a sequence which uses a BDT entry in which the length field is positive or zero. The engine will generate an event queue entry for the local DMA Context to indicate the BDT entry fault, and will not perform any command string associated with the successful completion of the command.

On the transmit side, a segment is permitted to wrap off the end of a buffer descriptor and into the next; this is not allowed on the receive side.

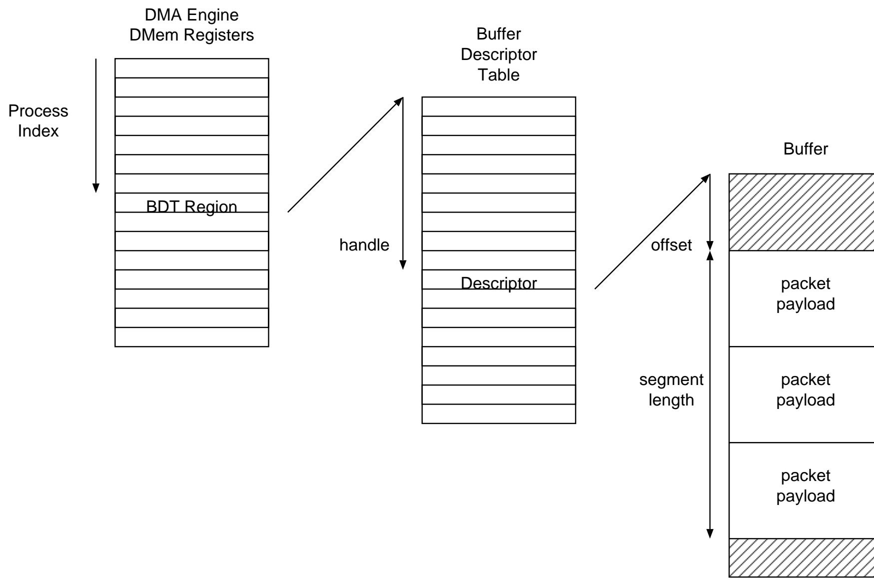
The physical address specified by a buffer descriptor must be aligned to a 64-byte boundary (low 6 bits zero).

Bit 0 of a BDT entry may be set to 1 to indicate that the buffer is read-only; use of such an entry for a receive buffer will cause a bdtFault.

Buffer Descriptor Class
DmaBufferDesc
Attributes
-kernel

Bit	Mnemonic	Type	Definition
d0[35:0]	physAddress		Physical address of start of buffer (address must be 64-byte aligned)
d0[63:36]	len		Length of physically-contiguous region

Figure 4.2: Buffer Addressing



Bits 5:1 of physAddress must be zero to ensure 64-byte alignment of data references. Bit 0 is not interpreted as an address bit, but if set restricts the buffer to read access only (DMA transfers cannot write; bdtFault is reported instead).

It is the current plan to make all BDT entries describe one page of memory - 64KBytes. This is large enough for efficient segment transfer and makes the VM management problem much easier.

4.7.4.1 Virtual Memory swapping

The user processes which depend on the DMA engine for communication services are ordinary Linux processes. As such, some pages of their virtual address space may not be in memory. Many interprocessor communication systems deal with this problem by requiring pages with active buffers to be "pinned", so that they cannot be paged out. This requires an explicit system call to pin and un-pin the buffers, or constrains the program to fit in the available memory and keep the entire data space pinned. We have chosen instead to assume that active buffers are in physical memory, and provide an escape mechanism for the rare cases in which that fails.

When the kernel in any SMP decides to swap out a page, it has to ensure that all processors have invalidated the page entry in their TLBs; in our system, it must also invalidate any corresponding entries in the BDT, and invalidate the BD cache in the hardware.

4.7.5 Route descriptor

The Route Descriptor Table (RDT) contains routing directives to get from this node to a specific Unix process on another node, typically by three disjoint paths. Route descriptors are protected from modification by the user; they are accessed by handles like buffer descriptors. A Route Handle, which appears in command queue entries, is a 28-bit unsigned integer less than the RDT size; it identifies a specific RDT entry as an offset relative to the RDT region.

Each process has a register representing the starting physical address and length of the route descriptor table (RDT) for that process (see Table **??**). It is a software decision whether RDT's are shared among processes. Each RDT entry is 8 bytes: 32 bits of routing directives, 4 bits of starting virtual channel number, a 16-bit process id on the destination node, and a 4-bit index which identifies the hardware process associated with the destination process id. The Route Descriptor also contains a 2-bit field identifying the output port associated with a path, so that a command using it can be stored on the appropriate transmit port queue.

All packets are given a path to their destination node and process at the time a command is enqueued in the source node's DMA Engine. The path is described by a string of routing directives, one per switch, indicating the output port to use on that switch. After selecting the output, each switch shifts the routing directive right two bits, discarding one directive and exposing the next for use at the next switch. Upon arrival at the destination

node, the process id in the packet is compared against that of the context selected by process index to determine the context in which the packet should be treated.

Route Descriptor Class
RouteDescriptor
Attributes
-kernel

Bit	Mnemonic	Type	Definition
d0[1:0]	txPort		Output port used for this path
d0[11:8]	virtChan		Initial virtual channel
d0[15:12]	processIndex		Remote process index
d0[31:16]	processID		Remote process id
d0[63:32]	path		Routing string for switch fabric - shift right each hop

Descriptor Cache The DMA engine caches up to 128 route descriptors and buffer descriptors. Any time that software modifies the RDT or BDT, it must write the corresponding handle to routeHdlPrefetch or bufferHdl-Prefetch, respectively, in the DmaAppIface1 for the corresponding context to keep the cache coherent with the table in memory.

Broadcast We considered creating a broadcast mechanism in the switch, so that a broadcast packet received on any input port would be replicated on all the outputs, until a time-to-live counter expired. We abandoned that approach for several reasons:

- While it works extremely well in a perfect Kautz graph, it becomes very messy if there are any dead links or nodes in the graph, or if there are non-Kautz topologies in the system.
- The packet contents must be the same everywhere, so there is no way to individually identify the target process(es). As a result, each node must decide whether there is any appropriate target process for each broadcast message.
- The requirement to replicate a packet to all output ports significantly complicated the switch design, which associates each packet buffer with an input/output crosspoint.

Instead, the DMA engine has provision for accepting a command (Do_Cmd) which directs the transmission of several output packets to software-selected destinations, allowing the construction of multicast trees with software-selected fanout, targeting specific processes at each destination node, and creating no new requirements for the switch fabric.

4.7.6 Heap

There are a number of data structures shared between the DMA engine hardware and the library software, which may be running on an application processor or the fabric processor; those structures need to be accessible to both, but the hardware uses physical addresses, while the software uses virtual addresses. To resolve this difference, we use a region of memory (called the *Heap*) which is user-writable and contiguous in both physical and virtual address spaces, and we refer to objects in that space by means of offsets (*handles*) within the heap. Such objects include communicators, the temporary values and fanout commands used by barriers and collectives, and unexpected eager messages.

Objects in the heap are referenced by handles, which are checked against the size of the heap, which is controlled by the kernel. A handle which exceeds the size of the heap results in a heap handle failure, which will be reported on the event queue of the local process.

Reserved Heap for Notifiers The first 8K bytes of the heap (addresses below 0x2000) are reserved for use by the DMA Engine, and must be zero at initialization time; microcode uses bits in that area to record DMA receiver buffer descriptor faults until they have been reported on the event queue.

4.7.7 Protected data structures

The hardware presents three interface pages for each of the 14 DMA Contexts it services (one writable by user mode, one user readable and writeable, and one writable only by the kernel). In addition, the kernel has direct read-write access to other information stored in the DMA Engine DMEM. Access control is managed by the processor's virtual address translation hardware.

It is a choice for software whether and how to use the kernel processes defined by the hardware, but the option is available to assign one to each processor on a node, so that packets and interrupts can be delivered to a dedicated processor (a fabric processor, for example) rather than an interrupt routine on one processor which might then have to notify the scheduler on another, for example. The hardware simply presents the pages as described in Table **??** – software may map them as needed. [Additional process id's might be useful if we want to deschedule a process while it communicates or while a priority process runs.] [Offsets are more or less arbitrary, and may change.] XXX NOTE the DmaProcCtlStatus page is actually just process specific storage cells in DMEM, refer to the DMEM map. The Write-Only items in DmaAppIFace are addressed by stores through the DMA External I/O addresses in RA_DmaAppIface1 + (ProcessIndex * 0x10000) + offset {where the L2 cache hardware converts them to SPCL operations on the CSW}. The RO and RW items are addressed through RA_DmaAppIface0 + (ProcessIndex * 0x10000) + offset using load (RDIO) and store (WTIO) operations.

Class

DmaProcCtlStatus

Attributes

-kernel

Bit	Mnemonic	Type	(Kernel Access)	Definition
d0[31:16]	processID		KRW	16-bit process id, unique within node
d1[63:0]	counters		KRW	Sixteen 4-bit counters for use by collectives
d2[63:0]	cmdQuota		KRW	Max queued commands for this process, minus 1
d3[63:0]	deferredCnt		KRW	Neg number of remote commands deferred by quota
d4[63:0]	eventQRegion		KRW	Region containing Process Event Queue
d5[63:0]	eventQRdPtr		KRW	Event Queue Read (head) pointer
d6[63:0]	eventQWrPtr		KRW	Event Queue Write (tail) pointer
d7[63:0]	heapRegion		KRW	Region containing Library Heap
d8[63:0]	cmdQRegion		KRW	Region containing Process Command Queue
d9[63:0]	cmdQRdPtr		KRW	Command Queue Read (head) pointer
d10[63:0]	cmdQWrPtr		KRW	Command Queue Write (tail) pointer
d11[63:0]	bdtRegion		KRW	Region containing Buffer Descriptor Table
d12[63:0]	rdtRegion		KRW	Region containing Route Descriptor Table
d13[11:0]	eventIntCause		KRW	Interrupt cause code when event is queued
d13[15:12]	eventIntTarget		KRW	Bus stop number to which interrupt is delivered

[Doublewords d4 through d12 are of type DmaBufferDesc.]

Class

DmaAppIface1

Attributes

Bit	Mnemonic	(User Access)	Definition
d0[63:0]	eventQRdSize	WO	Written by application to indicate size (in bytes) of item taken from event queue
d1[63:0]	cmdQWrSize	WO	Written by application to indicate size (in bytes) of new commands
d2[63:0]	routeHdlPrefetch	WO	Written by application with an RDT handle to preload or invalidate the route cache entry at that offset
d3[63:0]	bufferHdlPrefetch	WO	Written by application with a BDT handle to preload or invalidate the buffer descriptor cache entry at that offset

Software must write routeHdlPrefetch or bufferHdlPrefetch following any change to the RDT or BDT, respectively, to ensure that the change is recognized by the Dma Engine. The value written is the offset in the RDT or BDT of the updated entry. If the offset exceeds the size of the selected table, no update occurs, and the Dma Engine increments qmgrErrorCnt.

Class

DmaAppIface0

Attributes

Bit	Mnemonic	(User Access)	Definition
d0[63:0]	eventQRdOffset	RO	Event queue read pointer offset within region
d1[63:0]	eventQWrOffset	RO	Event queue write pointer offset within region
d2[63:0]	cmdQRdOffset	RO	Command queue read pointer offset within region
d3[63:0]	cmdQWrOffset	RO	Command queue write pointer offset within region
d4[63:0]	fastCmdHdr	WO	Header doubleword of Send_Event or Send_Cmd for fast launch; a copy of header on command queue

Restriction Registers in DmaAppIface0 and DmaAppIface1 must not be accessed while the DMA Engine has any threads disabled, or while the countdownHalt bit is set. Doing so can hang the processor. [The restriction on disabled threads does not currently apply, because we do not use mutex locking in the ioAccess microcode thread.]

The value written to fastCmdHdr must be the same as the header doubleword of the next command on the command queue, otherwise operation of the command is unpredictable.

Command Queue The command queue is written into memory by software. The first word of the command contains the payload length, by which the hardware can know how many bytes to read to complete the command. The DMA engine copies it either directly to the appropriate port, or to the appropriate queue for the required port, where it will be serviced in order. The length of every queue entry is always 128 bytes, which need not all be written by the processor.

Application software has two means by which it can notify the DMA engine of a new command on the command queue:

- By writing a multiple (N) times 128 to cmdQWrSize, software indicates that N new commands have been added to the queue.
- By writing the header doubleword of a command to fastCmdHdr, software indicates that one new command has been added to the queue. This function works only for SEND_EVENT, SEND_CMD, and PUT_IM_HP commands, and requires that the txPort field in the header is set correctly. In typical circumstances, this mechanism allows lower-latency processing of the command.

Command Quota The kernel assigns a quota to each process for the number of commands that it may have in the port queues at any time. Both local and remote commands are charged against that quota. When the quota is reached, the DMA engine stops accepting commands from the process command queue, and any received commands are copied to the event queue rather than a port queue. Any time a received command is sent to the event queue, the deferred count is incremented, and all further received commands are sent to the event queue until the deferred count returns to zero. [This is to keep received command processing in order] Software must set bit 16 in the header of any deferred command in the command queue, so that the DMA engine knows to adjust the deferred count.

The value in the *cmdQuota* register should be initialized to one less than the maximum number of outstanding commands allowed to the process; zero indicates that the process is allowed only one command at a time.

Do_Cmd can execute a string of commands. Once that string is started (implying that cmdQuota is positive), it is enqueued in its entirety, even if doing so drives cmdQuota below zero. Therefore, the port queues must be sized to accomodate a number of commands at least equal to:

[(cmdQuota + (execLimit/128)) * number_of_processes]

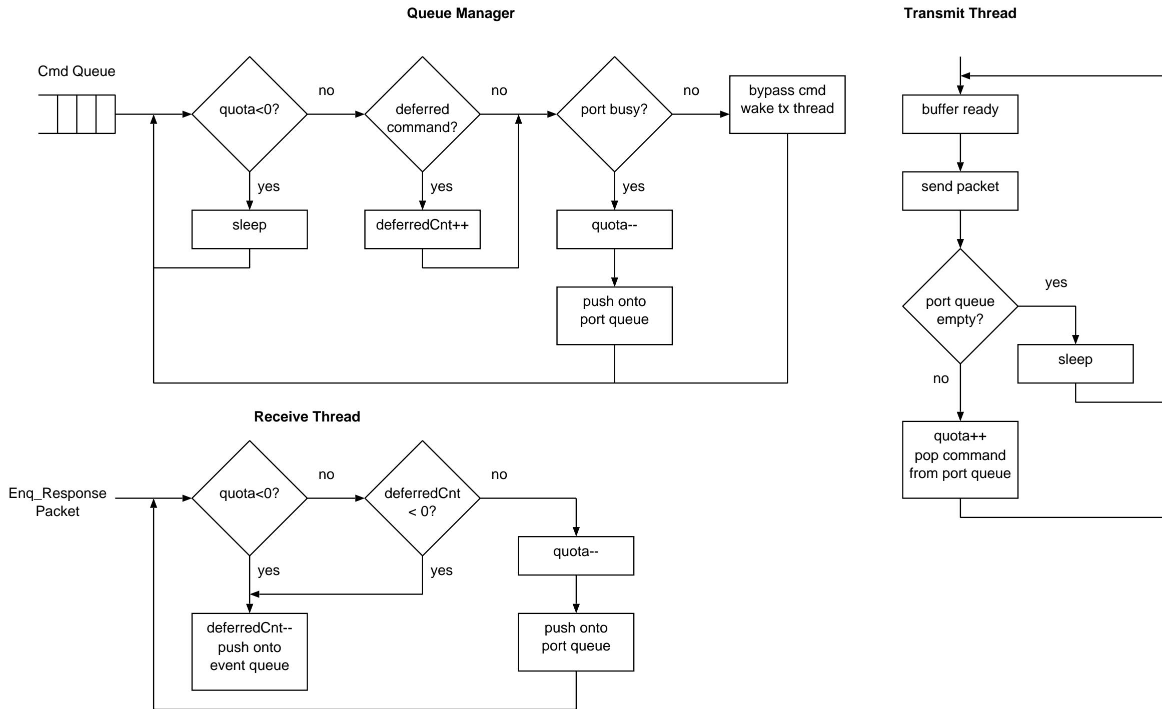
Figure 4.3 outlines the treatment of command quotas and the deferred count.

Interrupt Cause Register 13 (EventIntCause and EventIntTarget) will not cause an interrupt if zero. Bits 11:8 select an interrupt cause register at the processor selected by EventIntTarget, and bits 7:0 overwrite any interrupt cause value previously in that register. Because there are only 8 interrupt cause registers per processor, bit 11 must be zero.

4.7.8 DMA Engine Common Control/Status

The Common Control/Status variables, the contents of which are listed in Table **??**, are used by the DMA engine to manage the transmit queues for each output link. These values are typically initialized at boot time and otherwise ignored by software.

Figure 4.3: Command Quota and Deferred Count



Each queue is described by three doublewords, the first of which specifies the physical address and length in bytes of the memory used by the queue; the second contains a write (tail) address, and the third a read (head) address. These three doublewords are used directly by the DMA engine, but are not accessible to the application; the application sees only offsets from the beginning of the queue region, so it is unaware of relocation of the queue by the operating system when the process is paged out and back in.

Note everywhere a DmaQDesc occurs, the address is a byte address and the length is the negative byte length.

Software should refer to these variables through the names, as defined in the dma.load file; the assignments to specific dmem offsets are subject to change.

Class

DmaQDesc

Attributes

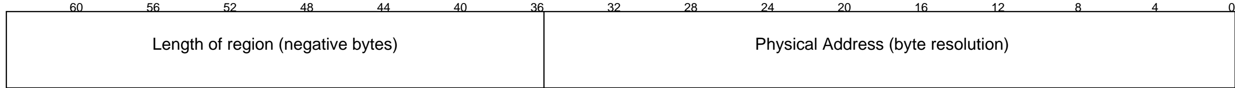
-kernel

Mnemonic	Bit	Definition
physAddr	d0[35:0]	Queue region physical address
len	d0[63:36]	Queue region negative length in bytes
wrAddr	d1[35:0]	Queue write (tail) address
wrLen	d1[63:36]	Queue write negative length hint
rdAddr	d2[35:0]	Queue read (head) address
rdLen	d2[63:36]	Queue read negative length hint

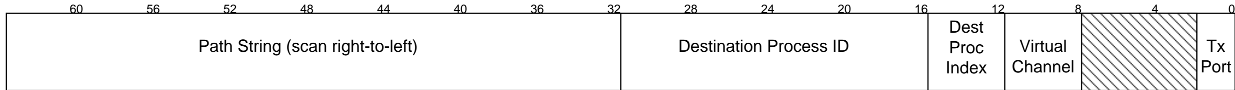
Symbol	Dmem Offset	Type	Definition
QmgrErrorCnt	0xe78		Count of commands ignored
ExecLimit	0xf78		Max allowed length of Do_Cmd string (in bytes)
UcodeVersion	0xff8		microcode version number
PortQRegion	44*16+port+bg*8		Queue region physical address and length
PortQRdPtr	45*16+port+bg*8		Current read (head) pointer for each port queue
PortQWrPtr	46*16+port+bg*8		Current write (tail) pointer for each port queue

Figure 4.4: Data Formats

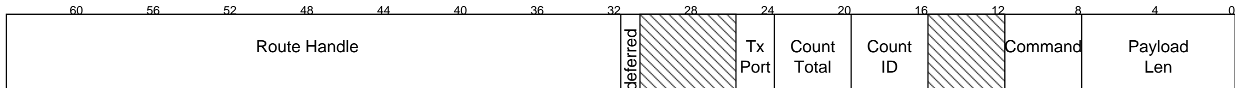
Buffer Descriptor, Queue Region, Heap, BDT, & RDT Descriptors



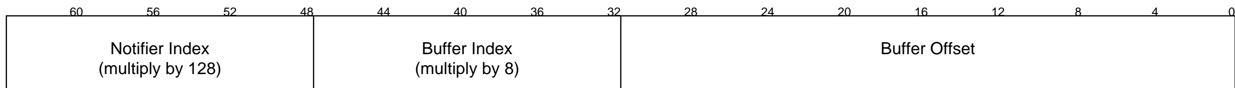
Route Descriptor



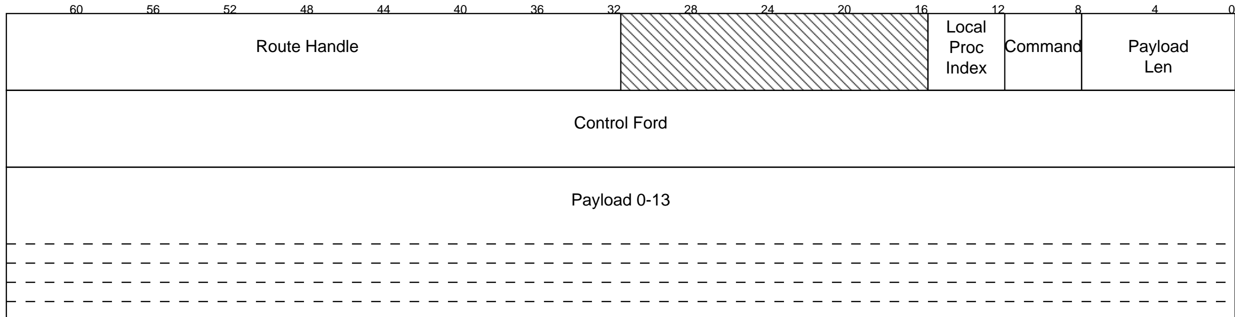
Command



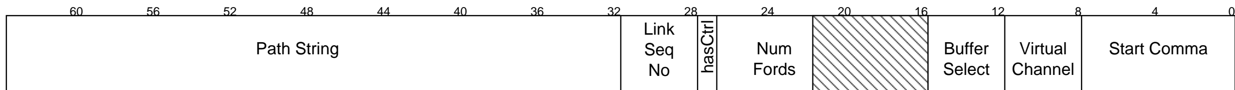
DMA Src/Dst Control



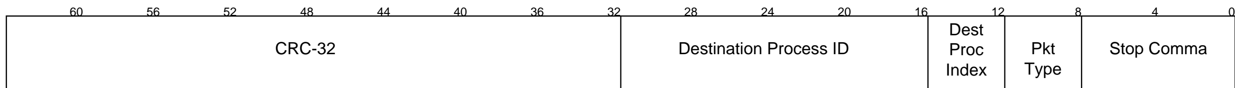
Port Queue Entry



Packet Header



Packet Trailer



4.8 Commands

The DMA engine receives commands as blocks of data, either written by a processor on the same node, or as the payload of a received Enq_Response packet. In either case, bits 63:32 of the first doubleword of the block are interpreted as a route handle, which is evaluated in the RDT of the current process to determine what queue and path to use for the packet. The RDT entry determines the port that will be used by the command, and hence the appropriate queue for holding the command until the port is available.

Enum
DmaCmdType
Attributes
-kernel

Constant	Mnemonic	(Queue)	(Packet)	Definition
4'd0	NOP	none	none	No operation
4'd2	SUPERVISE	varies	none	Supervisory and management functions
4'd3	SEND_EVENT	Tx_fg	Enq_Direct	Deliver data directly to remote event queue
4'd5	PUT_BF_BF	Tx_bg	DMA	Transmit segment from local buffer to remote buffer
4'd8	DO_CMD	Tx_fg	any	Enable commands found in local heap
4'd9	SEND_CMD	Tx_fg	Enq_Response	Enqueue payload on remote command queue
4'd10	PUT_IM_HP	Tx_fg	Wr_Heap	Transmit command data to remote heap

Command encoding is chosen to make command codes match the packet types they send, insofar as possible, with the valid packet types all coded with even parity (probably not necessary, but we still have plenty of code space...).

Software must inform the DMA Engine of any new command by writing that command to the next available 128-byte block of the command queue, and either:

- (Standard method) Write the number of new commands times 128 to the I/O register called cmdQWrSize, or
- (fast path for one command only) Write the header of the new command to the I/O register called fastCmdHdr.

4.8.1 Command Header

The first doubleword of every command has a uniform structure, shown here:

Class
DmaCmdHead
Attributes
-kernel

Bit	Mnemonic	Type	Definition
d0[7:0]	len		Immediate payload length in bytes: 0-112
d0[11:8]	cmdtype	DmaCmdType	Command type code
d0[15:12]	pidx		Reserved for process index
d0[19:16]	countId		Do_Cmd counter selector
d0[23:20]	countTotal		Do_Cmd counter reset value
d0[25:24]	txPort		Output port (hint) to be used for command
d0[30:26]	reserved		Reserved; must be zero
d0[31]	deferred		Set to indicate deferred remote command
d0[63:32]	routeHandle		Route handle for path to destination

4.8.2 Send_Event Command

The Send_Event command instructs the DMA engine to create and send an Enq_Direct packet, whose payload will be stored on the event queue at the destination process. If it isn't processed immediately, a Send_Event command waits on the Tx_fg (foreground) queue.

Class
DmaCmdEvent
Attributes

Bit	Mnemonic	Type	Definition
d0[63:0]	header		Command type is Send_Event
d1[63:0]	control		Reserved; must be zero
d2[63:0]	payload		First payload data

The length field in byte 0 of the header encodes the payload length, which must be a multiple of 8 and between 8 and 112 bytes.

Software may optionally use the "fastCmd" mechanism to perform a Send_Event command, saving significant overhead if the required output port is idle.

4.8.3 Send_Cmd Command

The Send_Cmd command instructs the DMA engine to create an Enq_Response packet, with a payload to be processed as a command at the destination node. If it isn't processed immediately at the source, the Send_Cmd command waits on the source Tx_fg queue. The Send_Cmd command contains a nested command as its payload; that nested command will be interpreted at the remote node as if it had been issued by the receiving process, but the nested command must not be Send_Cmd or Supervise. The nested command in the Enq_Response packet determines which command queue (Tx_fg or Tx_bg) at the remote node receives the command; the route handle in the packet payload determines the RDT entry selected, and thus the output port selected at the destination.

Class

DmaCmdSendCmd

Attributes

Bit	Mnemonic	Type	Definition
d0[63:0]	header		Command type is Send_Cmd
d1[63:0]	control		Reserved; must be zero
d2[63:0]	payloadHead		Payload, a nested command to be enqueued at the receiver
d3[63:0]	payloadCtl		Payload; control word of nested command
d4[63:0]	payloadPay1		Payload of nested command
d5[63:0]	payloadPay2		Payload of nested command
d6[63:0]	payloadPay3		Payload of nested command
d7[63:0]	payloadPay4		Payload of nested command
d8[63:0]	payloadPay5		Payload of nested command
d9[63:0]	payloadPay6		Payload of nested command
d10[63:0]	payloadPay7		Payload of nested command
d11[63:0]	payloadPay8		Payload of nested command
d12[63:0]	payloadPay9		Payload of nested command
d13[63:0]	payloadPay10		Payload of nested command
d14[63:0]	payloadPay11		Payload of nested command
d15[63:0]	payloadPay12		Nested command payload continues up to 12 doublewords

The length field in the header is variable; it gives the length of the nested command, including its header and control word. The header of the nested command also has a length field which can be at most 96 bytes.

Queueing of the nested command at the destination is controlled by the *cmdQuota* and *deferredCnt* process variables at the destination. If the *deferredCnt* is non-zero, or the remaining *cmdQuota* is negative, the nested command is pushed onto the event queue with an event type that indicates it is a deferred command, and the *deferredCnt* variable is incremented. Otherwise, the command is processed as if it had been pushed onto the destination node's command queue by software on that node, and the quota is decremented.

Software may optionally use the "fastCmd" mechanism to perform a Send_Cmd command, saving significant overhead if the required output port is idle.

4.8.4 Do_Cmd Command

The Do_Cmd command instructs the DMA engine to perform a string of commands which will be found in the local heap. The string of commands must not include Do_Cmd commands. Each command in a string contains its own route handle and command code, which together determine the queue on which that command waits.

Class

DmaCmdExecute

Attributes

Bit	Mnemonic	Type	Definition
d0[11:0]	header		Command type is Do_Cmd
d0[19:16]	countId		Counter selector
d0[23:20]	countTotal		Counter reset value
d1[31:0]	execHandle		Heap handle for first command
d1[63:32]	execCount		Number of bytes in command string

The countId field identifies one of 16 4-bit counters associated with the target process; Do_Cmd decrements that counter. If the starting value of the counter is zero, the value is replaced by the contents of the countTotal field and the commands specified by execHandle are enqueued. If the starting value is non-zero, the decremented count is saved and the specified commands are ignored.

The counters for each process are in the *counters* register in DmaProcCtlStatus; the counter selected by countId=0 is in bits 3:0 of that register; the counter selected by countId=15 is in bits 63:60. Counter 0 may be implicitly accessed by the successful completion of a Put_Bf_Bf command; it is ordinarily left containing 0.

The execHandle is a byte offset in the heap at which the first command will be found; execCount is the number of bytes of the commands, and each command is 128 bytes long. ExecCount must not exceed the value in the *ExecLimit* register, controlled by the kernel. When Do_Cmd executes a command string, all the commands are enqueued, and *cmdQuota* is decremented for each, regardless of the sign of *cmdQuota*; the port queues must therefore be sized to allow *ExecLimit* space after *cmdQuota* is exhausted.

Do_Cmd executes foreground commands for each transmit port in the order specified in the command string, but there is no order guarantee with respect to the command queue, background commands, or other ports.

Unused fields in the header word (length and route handle) must be zero.

Do_Cmd must not be issued on the "fast path"; doing so results in a cmdFault.

4.8.5 Put_Bf_Bf Command

Put_Bf_Bf commands instruct the DMA engine to create and send a sequence of DMA packets to the remote node; the packet payload is taken from a buffer identified by a buffer handle. Put_Bf_Bf commands wait on the Tx_Bg (background) queue for the availability of a transmit context. The implication is that while most commands following any given route are completed in the order in which they were enqueued by software, Put_Bf_Bf commands may be delayed with respect to other commands to the same destination. Put_Bf_Bf will never be started before completion of previously-queued commands which use the same route.

Class

DmaCmdPutBf

Attributes

Bit	Mnemonic	Type	Definition
d0[63:0]	header		Command type is Put_Bf_Bf, length 32
d1[31:0]	segLength		Segment length in bytes
d1[63:32]	execRouteHandle		Route handle to notify of successful completion, or zero if local
d2[31:0]	txOffset		Byte offset from local buffer descriptor base
d2[47:32]	txBufferHandle		Local Buffer Descriptor index
d2[63:48]	txNotifier		Segment identifier for transmitter
d3[31:0]	rxOffset		Byte offset from remote buffer descriptor base
d3[47:32]	rxBufferHandle		Remote Buffer Descriptor index
d3[63:48]	rxNotifier		Segment identifier for receiver
d4[63:0]	swBucket		Available space for data delivered to receiver event queue
d5[31:0]	execHandle		Heap offset of receive-completion command string
d5[63:32]	execCount		Length of receive-completion command string

TxOffset and RxOffset define the starting byte address of the destination and source buffers with respect to buffer descriptors selected by txBufferHandle and rxBufferHandle on the remote and local nodes, respectively. The calculation works as follows:

The txBufferHandle is extracted and multiplied by 8; the result is added to the sending process BdtRegion pointer, where the source buffer descriptor is found. The txOffset (which must be a multiple of 4) is added to the address in the buffer descriptor to give the starting address of the source buffer.

At the receiver, a similar process interprets the rxBufferHandle and rxOffset in the destination process environment, except that the rxOffset must be a multiple of 32.

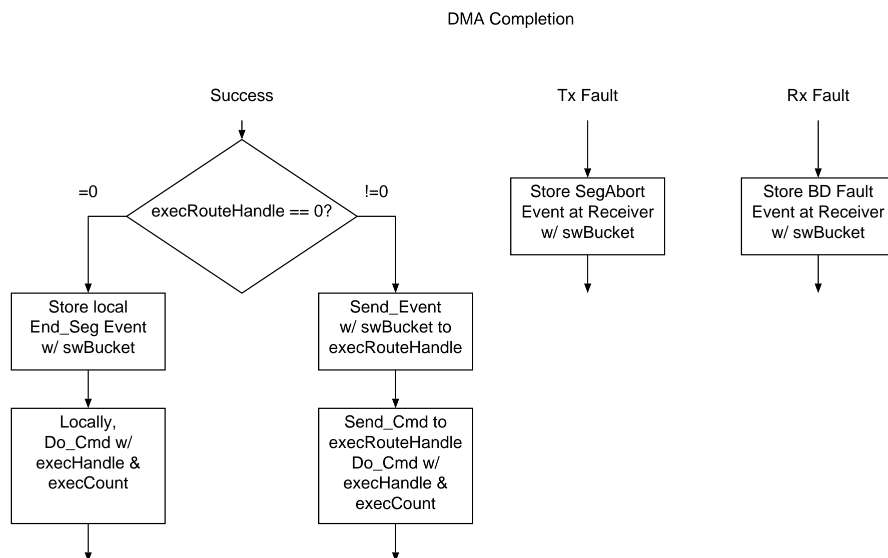
In the event of a buffer descriptor fault (a buffer handle is too large, implying a buffer descriptor outside the bdt region, or a buffer descriptor with length of zero) the transfer is terminated, and an BdFault (rx) or SegAbort (tx) event is stored on the event queue at the receiver. There is no direct notification of the transmitter, even if the fault occurs there.

On the transmit side, the offset plus length of a segment may be allowed to exceed the maximum address implied by the Tx Buffer Descriptor. In that event, the transmitted data runs to the end of the specified region, then any excess comes from the region specified by the next buffer descriptor. No such continuation is permitted at the destination.

Upon successful completion of a segment, the receiver tests the execRouteHandle. If zero, the receiver executes a string of commands, as described by the execCount and execHandle (same as Do_Cmd). If non-zero, the receiver builds a Do_Cmd containing the execCount and execHandle, and sends it as if it were in a Send_Cmd with that execRouteHandle as the route.

Note to software implementors: Library software must be prepared to deal with source and destination buffers which may have different alignment. The hardware is designed to handle the most common cases, but there are several conditions which require special handling by software:

- If the destination buffer does not start and end at cache block (32-byte) boundaries, software must use another mechanism (probably built upon Send_Event) to deliver the data which belongs in partial blocks.
- If the destination buffer does not start at a 64-byte boundary, the transfer will make most efficient use of the memory bandwidth if the library uses a short segment to achieve 64-byte alignment for the bulk of the transfer.
- If the source and destination buffers do not have the same alignment, the starting offset in the source buffer should be specified to align the first packet to a 64-byte boundary at the destination.
- If the source and destination buffer alignments differ by an amount which is not a multiple of 4, the alignment must be adjusted by a software copy before or after the transfer.



4.8.6 Put_Im_Hp Command

Put_Im_Hp commands instruct the DMA engine to send a packet to the remote node; the packet payload comes directly from the command and is written to the remote heap. Put_Im_Hp commands wait on the Tx_fg (foreground) queue for the availability of an output port.

Put_Im_Hp sends a single Wr_Heap packet, whose payload comes directly from the command and is written to the remote heap.

Class

DmaCmdPutImHp

Attributes

Bit	Mnemonic	Type	Definition
d0[63:0]	header		Command type Put_Im_Hp
d1[31:0]	heapHandle		Heap offset at destination (aligned 64)
d2[63:0]	payload		Initial payload doubleword (first of up to 14)

The length field in byte 0 of the header gives the length of the payload in bytes. It must be a multiple of 8, and the payload will be extended with zeros to the next 32-byte boundary when it is stored at the destination.

Software may optionally use the "fastCmd" mechanism to perform a Put_Im_Hp command, saving significant overhead if the required output port is idle.

4.8.7 Supervise Command

The Supervise command provides control mechanisms for management of the DMA engine. It serves as a marker which writes its payload to the local event queue when all earlier foreground commands for a selected port have been sent. The marker is intended to provide library software with a reliable indication that space in the command queue and/or heap is available.

Note that completion of DMA transfers is generally reported by an endSeg event; Supervise is useful for flushing the commands in the transmit foreground queues.

Class

DmaCmdSupervise

Attributes

Bit	Mnemonic	Type	Definition
d0[63:0]	header		Command type is Supervise, length 16. Port to mark is specified by txPort 25:24
d1[63:0]	control		Reserved
d2[63:0]	payload		First payload data doubleword, copied to Event queue d0
d3[63:0]	payload1		Second payload data doubleword, copied to Event queue d1

The Supervise command selects an output port using bits 25:24 of the header, and stores its payload on the local event queue after processing all earlier commands for the same output port.

Supervise evaluates d0[63:32] as a route handle, like other commands, even though it will be used only to verify that the port selected by the header matches that selected by the route.

Supervise may not be nested inside Send_Cmd.

Software may optionally use the "fastCmd" mechanism to perform a Supervise command, saving significant overhead if the required output port is idle.

4.8.8 Undefined Commands

Command codes which have not been defined otherwise result in a cmdFault event being stored on the event queue of the context in which they occur.

4.9 Packet formats

4.9.1 Packet header and check

Packet sizes are multiples of 8 bytes, so that packet boundaries correspond to symbol framing boundaries on the link. Each 8-byte unit is referred to as a "ford"; see Matt for derivation and justification. Data packets consist of four or more fords, up to 19. The first ford of every data packet (the *header*) contains a routing string, a virtual channel number, a buffer index for the next switch, and a link sequence number for error recovery; the second ford, called the *control word*, is interpreted by the receiving DMA engine to control where and how the payload is stored; the last ford, the *trailer*, contains the packet type, a 20-bit identification code for the target process at the destination node, a CRC checksum, and 8 constant bits (which are the translation of the "comma" symbol used to mark the end of the packet). See Table 4.2. The control word may or may not be present; the *hasCtl* flag in the header is set if and only if the word is present.

Idle packets consist of a single ford marked by a comma symbol which is used only by Idle. The remaining bits may be used for diagnostic or out-of-band information and a CRC checksum.

Table 4.2: Packet Header and Trailer

Field	Bits	Source	Definition
Start of Packet	7:0	Switch	Start Comma
Virtual Channel	11:8	RDT	Current arbitration level
Buffer Select	15:12	Switch	Next hop target buffer
	21:16		Reserved
NumFords	26:22	Switch	Length of packet in fords
HasCtl	27	DMA	Set to interpret second ford as control
Link Seq No	31:28	Switch	Packet seq no on this link
Route	63:32	RDT	16 2-bit routing instructions
End of Packet	7:0	Switch	Terminating Comma
Packet Type	11:8	CmdQ	Controls Receiver Processing
Process Index	15:12	RDT	Select Control/Status page
Process ID	31:16	RDT	Match Unix PID in CS page
CRC	63:32	Switch	Error Detection for whole packet

Non-idle packets need a type field, to control their interpretation, and a process id, which must match that assigned to the receiver by the kernel. This is to prevent confusion when processes are rescheduled or moved between processors, and to prevent rogue processes from examining or modifying unrelated process memory.

[We still need to define any required debug and performance monitoring features.]

4.9.2 Packet Types

Table 4.9.2 lists the defined packet type codes. Any packet received with an undefined type is reported as an error and discarded. [Currently, all valid packet types have even parity, to make it that much more difficult to mistake a corrupted packet. Next we should use the odd-parity codes which are distance 3 from poison. Seems excessive, at this point, but we have plenty of codes still.]

Enum

DmaPktType

Constant	Mnemonic	Definition
4'b0011	ENQ_DIRECT	Push packet payload onto software event queue
4'b0101	DMA	Store payload according to receive context
4'b1001	ENQ_RESPONSE	Push packet payload onto receiver's command queue
4'b1010	WR_HEAP	Store payload into process heap
4'b1100	DMA_END	Signals end of DMA segment
4'b1111	POISON	Discard packet

4.9.3 Direct Transmission: Enq_Direct

Short messages, consisting of one or a few packets, are sent by the sending process constructing a command with a route handle and the contents of the desired packet, whose payload is deposited on the event queue of the receiving process. The event queue is processed by software at the receiver.

Table 4.3 shows the form of a packet whose contents will be deposited on the event queue for processing by software; similar packets are available to store to the DMA Engine's command queue. Another form stores to the heap, using the control doubleword to specify a heap offset.

Enq_Direct packets are generated by Event commands.

Event queue entries are all 128 bytes. The DMA engine writes the packet payload, and fills to the next 64-byte boundary with zeros.

4.9.4 DMA

DMA packets are the heavy truckers of the SiCortex fabric. They carry the high-volume message traffic between cooperating nodes which have set up matching transmit and receive contexts. In addition to the payload and the

Table 4.3: Direct Queue Packet Fields

Field	Size (bytes)	Source	Comments
Header	8	DMA Engine	As defined in Table4.2
Control	0		Skipped; hasCtl=0
Payload	8-112	CmdQ	For use by software
Trailer	8	DMA Engine	As defined in Table 4.2

header/checksum overhead carried by all packets, DMA packets carry a control ford which tells the receiver's DMA Engine where to store the payload in the destination buffer. The format of the control ford is shown below:

Class

DmaCmdCtl

Attributes

Bit	Mnemonic	Definition
d0[31:0]	offset	Byte offset of packet payload with respect to buffer descriptor
d0[47:32]	bufferHandle	Index into BDT for buffer descriptor (multiply by 8)
d0[63:48]	notifier	Bit index into heap for error flag

See Table 4.4.

Table 4.4: DMA packet fields

Field	Size (bytes)	Source	Comments
Header	8	DMA Engine	As defined in Table 4.2
Control	8	DMA Engine	As defined above, in class DmaCmdCtl
Payload	8-128	Buffer	User data
Trailer	8	DMA Engine	As defined in Table 4.2

Message buffers are not necessarily aligned with respect to cache blocks, at either the transmitting or the receiving node, but the DMA engine requires that a received DMA packet must be aligned so that its payload starting address precisely corresponds to an integral number of L2 cache blocks (64-byte boundary). Therefore, the transmitting node's DMA engine may be required to form packets from up to three cache blocks, with alignment at any 4-byte boundary; library software is obliged to use Enq_Direct packets to pass data at the beginning and end of a message which do not align to a cache block boundary.

The DMA payload length is permitted to be less than a multiple of 32 bytes; in that event, the receiver will extend the payload with zeros to the next larger 32-byte boundary

When a DMA packet is received, the receiver uses the buffer handle (*8, for a byte address) to obtain a buffer descriptor from the process BDT. The buffer offset is added to the descriptor base address to obtain the address at which the payload is stored. In the event of a fault, the payload is not stored, and the microcode sets a flag in the heap (bit number rxNotifier mod 8 in byte rxNotifier / 64 of the heap). The flag is tested and cleared by a DMA_End packet when the transmitter finishes the segment; if set, the receiver stores a bdtFault rather than rxEndSeg.

DMA packets are generated by Put_Bf_Bf commands.

4.9.5 DMA_End

A DMA_End packet is sent following the final DMA packet of a segment to mark successful transmission. It contains sufficient information to allow the receiver to store an EndSeg or bdtFault event on the event queue, and if the transfer was successful, optionally activate a string of dependent commands at the receiver (if execRouteHandle is zero) or a remote node as specified by the execRouteHandle relative to the receiver's RDT.

4.9.6 Wr_Heap

Wr_Heap packets are used to write the Heap communication area allocated by the target process.

Under some circumstances, the sender of a short message may choose to use Wr_Heap packets to transfer the message data to the destination node before matching SEND with RECV, so that software at the destination can copy the data once a match has been made.

Table 4.5: DMA End packet fields

Field	Size (bytes)	Source	Comments
Header	8	DMA Engine	As defined in Table 4.2
Control	8	DMA Engine	Notifier and BD Handle as in DmaCmdCtl; execRouteHandle in 31:0
Payload0	8	Command	Software “bucket”
Payload1	8	Command	Exec Handle and Count
Trailer	8	DMA Engine	As defined in Table 4.2

Table 4.6: Wr_Heap packet fields

Field	Size (bytes)	Source	Comments
Header	8	DMA Engine	As defined in Table 4.2
Offset	8	CmdQ	Start offset within Heap
Payload	8-112	CmdQ	Data to be written to destination heap
Trailer	8	DMA Engine	As defined in Table 4.2

Offset must be a multiple of 64; length must be a multiple of 8. Writes to the heap always modify one to four aligned 32-byte blocks of memory. Memory beyond the last doubleword of payload is zeroed to the next 32-byte boundary.

Wr_Heap packets are generated by Put_Im_Hp commands.

4.9.7 Enq_Response

A Get request for a large message becomes an Enq_Response packet, created by the Initiator as part of a Receive command. When the initiator is ready to receive a segment, an Enq_Response is sent from the initiator to the responder (Table 4.7), containing a Put_Bf_Bf command to be used at the remote (responder) node. The command is processed by the DMA engine at the responder, subject to the same access constraints as if the entry had been placed on the command queue by local software.

Table 4.7: Enq_Response Packet fields

Field	Size (bytes)	Source	Comments
Header	8	DMA Engine	As defined in Table 4.2
Control	0		Skipped; hasCtl=0
Payload	16-112	CmdQ	Response command executed at destination
Trailer	8	DMA Engine	As defined in Table 4.2

Typically, an Enq_Response packet contains a Put_Bf_Bf command which directs transmission of a segment, but there are valid uses of other command types.

When an Enq_Response packet is received by a responder, the responder checks the cmdQuota and deferredCnt variables for the target process. If the cmdQuota is exhausted (negative) or the deferredCnt indicates there are previously-deferred commands awaiting service, the response command in the packet is pushed onto the target process event queue with code *deferredCmd*, and the deferredCnt process variable is adjusted. This is to prevent remote commands from overflowing the port queues. Library software associated with the process must recognize the deferred command and copy it to the command queue, setting bit 31 (*deferred*) in the header.

4.9.8 Poison

Poison packets are not intentionally generated by the DMA Engine, and are discarded when received. Any packet may be converted to a poison packet if some link along its path detects a CRC error. That link will request retransmission, but the corrupted packet may already have left the station, so the poison type code causes it to be ignored.

4.10 Notes on Complex Functions

4.10.1 Rendezvous

Rendezvous is the handshake sequence executed between a pair of processors planning to use DMA packets to pass a large message; it gives both participants the information needed to set up transmit and receive contexts.

Rendezvous is initiated by software injecting a rendezvous request as an Send_Event in the command queue. The request contains communicator, source rank, and tag. It also carries buffer alignment information. The initiating node sends the request to the responding node, where the DMA engine stores the packet in the event queue so that software can find a matching receive. Once the match is found, the responding node issues either a Put_Bf_Bf command or a Send_Cmd containing a Put_Bf_Bf, which produces a stream of packets. Upon completion of the segment transfer, the receiving node stores an endSeg event and (if successful) processes its completion command string.

There are substantial performance consequences from appropriate scheduling of segment transfers at Rendezvous; blindly queueing transfers in a FCFS order may result in severe hotspot congestion. It is up to software to reorder transfers for optimum performance.

4.10.2 Stride and Scatter/Gather

MPI specifies mechanisms by which the application can build messages that correspond to non-contiguous memory at the sender and/or receiver. The early plans for the DMA Engine included direct support for such messages, but they created a problem in that a packet which requires many main memory references may take much longer to service than its occupancy in any other stage of the communication pipeline; this creates the prospect of a message of such packets backing up the network in undesirable ways. Therefore, the fabric processor should be used for assembly and disassembly of non-contiguous messages, either by copying the data to and from contiguous buffers which are then transferred via rendezvous send/receive, or by transfer of convenient-sized chunks using directly-queued packets.

4.10.3 Barrier and Collective

A rough model: nodes in a communicator are organized in a tree (branching rate to be determined by experimentation) with a root, intermediates, and leaves. As each node reaches the collective operation:

- Leaf nodes send their contribution (using Wr_Heap packets; see Table 4.6) to pre-allocated heap cells in their immediate parent, an intermediate node.
- Intermediate and root nodes gather the contributions of their children and the local process. This can be in software, spin-waiting for completion, or using the counting facility in Do_Cmd to initiate transmission of the result toward the root.
- When the contributions from their leaves have all arrived, intermediate nodes send a group contribution to their parent (again using Wr_Heap packets).
- When the root receives all its contributions, it broadcasts the collective result to the entire communicator, using multicast.

Reduction operations which require arithmetic (sum, max) must defer to software for the arithmetic, but may choose to gather several layers of inputs through such a tree before invoking software to perform the reduction.

4.10.4 Multicast

The early design included a mechanism called Exploding Broadcast as part of the fabric switch; that approach has been abandoned for reasons outlined elsewhere. Current plans provide for a multicast mechanism in the DMA engine, implemented with ordinary point-to-point packets (carrying an Execute command) which can stimulate execution of multiple commands, sending output packets to software-selected destinations.

4.10.5 Out-of-band

The switch interface includes six pairs of registers corresponding to byte-wide send and receive paths to and from the immediate adjacent nodes on each of the switch input and output ports. Each register carries a byte of data plus a handshake bit. When a node writes its send register, the send register's handshake bit is cleared, and sets again after software in the remote node reads the corresponding receive register. The remote node's handshake bit is cleared when the byte arrives in the receive register, and sets when software reads the register. This mechanism is used by software in the early stages of configuring the fabric and booting the operating system, and remains available for any purpose required by software during normal operation.

4.10.6 Receive Matching

We looked for a way to match MPI_SEND with MPI_RECV in microcode, so that the rendezvous could be turned around without software intervention. We were unable to devise a satisfactory solution, and for the moment at least, it's not under consideration.

4.10.7 Initialization

This subsection will describe the process of initializing and starting the DMA engine in preparation for use, both at boot time and when a new process is allocated.

4.10.7.1 Black Hole

Upon power-up, the DMA engine, fabric switch, and links are in reset state, but there may be circumstances in which the initialization sequence is entered with some or all in operation. In particular, after a node crash induced by hardware or software failure, it is desirable to keep traffic flowing through the switch and links while the node reboots. To support such cases, the block reset register includes functions which ignore all packets entering or leaving the switch at its node.

4.10.7.2 Reset

During initialization, the following registers should be set up:

- the block reset register should be set to inhibit traffic into and out of the local interface of the fabric switch
- the thread select register should disable all 10 threads
- the ECC mode register should be set to enable correction
- the force error register should be cleared
- ECC error interrupts should be disabled in the interrupt mask register

After the instruction and data memories have been loaded and the common resources set up, these registers can be returned to their normal state.

4.10.7.3 Microcode load

The DMA Engine microcode assembler, dmaas, translates a symbolic representation of the microcode (called dma.lisp) into a numerical representation which specifies the microinstructions themselves and the initial states of dmem and thread-state variables. This is called the .load format:

4.10.7.4 Variable binding

Many of the DMA registers accessible through I/O reads and writes have values that are important to the device driver, but the particular address assignments may change from one version of microcode to another. The microcode .load format provides the necessary information to translate symbols to addresses, and initialization software is expected to refer to interface registers by using strings to name the register, translating the string to an I/O space address on the basis of the current .load file (perhaps using SymbolTableMap).

Microcode version By convention, a microcode variable named ucodeVersion is assigned to location dmem location 511 (0x1FF). It contains in bits 31:0 the svn revision number at which the source code was committed; in bits 39:32 an identification code for the API it implements (3 for this specification); and 63:40 are defined according to the API code.

4.10.7.5 Initialization of common resources

The dma initialization software which runs during the boot process loads the microinstruction memory (using writes to R_DmaImem), dmem constants and global variables (using writes to R_DmaDmem), and the thread state variables (using writes to R_DmaThreadPtr[] and R_DmaThreadPc[]) as specified in the .load file. The following table lists the symbols needed for system initialization:

Symbol	Index	Description (initial value)
portQRegion	0-7	Physical address and length of region reserved for transmit port queue
portQRdPtr	0-7	Transmit port queue read pointer (copy of portQRegion)
portQWrPtr	0-7	Transmit port queue write pointer (copy of portQRegion)
rxErrorCnt	-	Count of bad packets received
qmgrErrorCnt	-	Count of context 0 event queue overflows

Certain dmem values refer to physical memory regions which are allocated by the kernel for use by the DMA Engine. In addition to areas used by each process, each port relies on reserved memory regions in which it can store a queue. For each queue, there are three doublewords in dmem, called the region descriptor, the write pointer, and the read pointer; the region pointer and write pointer should be initialized to the same value: in bits 35:0, the physical memory address of the area of memory allocated for use by the queue (bits 5:0 must be zero); in bits 63:36, the negative length of the allocated region. Thus, if the allocated region is 65,536 bytes (0x10000) starting at address 0x123456780, the doubleword value should be 0xFFF0000123456780. The read pointer should have the same address in bits 35:0, but zero in 63:36.

The eight areas allocated for port queues must be non-overlapping, aligned to 128-byte boundaries, and a multiple of 128 bytes in length.

4.10.7.6 Initialization of process resources

As the system associates operating system processes to process state in the dma engine, it must allocate space in physical memory for the five communication regions used by each dma process: the heap, the buffer descriptor table, the route descriptor table, the command queue, and the event queue. The command and event queues are each described by dmem registers containing read pointer, write pointer, and region descriptor, as described above for the port queues. The heap, BDT, and RDT are each described by a single dmem register containing a region descriptor in which bits 35:0 contain the physical address of the start of the region, and bits 63:36 contain the negative of the region length. The following table lists the symbols needed for initialization of each process, whenever a new process binding occurs. To avoid an error wrap case, queue Rd and Wr pointers must be initialized to offset 128 in the region (add 128 to both the address and negative length fields).

Symbol	Description
processID	Process identifier (16 bits)
counters	Sixteen 4-bit counters used by Do_Cmd commands (init 0)
eventQRegion	Physical address and length of region reserved for event queue to this process
eventQRdPtr	Event queue read pointer (copy of eventQRegion)
eventQWrPtr	Event queue write pointer (copy of eventQRegion)
cmdQRegion	Physical address and length of region reserved for command queue from this process
cmdQRdPtr	Command queue read pointer (copy of cmdQRegion)
cmdQWrPtr	Command queue write pointer (copy of cmdQRegion)
BDTRegion	Physical address and length of region for buffer descriptor table
RDTRegion	Physical address and length of region for route descriptor table
HeapRegion	Physical address and length of region for process heap
cmdQuota	Number of concurrently queued commands available to this process, minus 1
deferredCnt	Number of remote commands currently deferred to event queue
eventIntCause	Interrupt cause word sent when an event is added to an empty event queue

4.10.8 Process Rundown

This subsection will describe the sequence of events required to deallocate a DMA Engine process.

4.11 Lessons for Next Time

4.11.1 Queue Manager

The performance of this design suffers from a couple of problems.

In the first place, the queue manager must read a command from memory, then translate its route, before knowing which tx thread will service it. And the requirement to keep commands in order makes it difficult to evaluate other commands during that process. A better design might require each software process to enqueue commands into separate queues for each port: 4 command queues per process.

In that model, each transmit thread could scan its own queues, executing fg commands as they were encountered, and pushing bg commands to the port queue, to be handled when all the fg commands were finished. This would make processing more efficient, both because of parallelism, and because control information would not need to be moved from memory buffer to dmem to packet buffer.

It would be necessary to come up with a way of pushing and processing commands received from remote nodes. They could have their own queue area in memory, treated like a separate process, or they could be pushed through the background port queue like DMA commands.

The “fastpath” mechanism points the right direction: use it for invocation of all locally-initiated commands (possibly except do_cmd). This allows dispatch to appropriate port thread right away, with RDT access before command fetch. In most cases, the queue access is needed only for the payload. The Tx thread might have separate priority levels for fastpath (nothing on queue), enqueue, foreground, and background. Received commands in Enq_Response packets would be enqueued by receiver.

4.11.2 Additional functionality

4.11.2.1 Enqueue/Dequeue commands

There should be a means by which a node can create a ring-buffer queue which is available to all processes in the same job to insert or remove entries; it may not be important to have more than one such queue per process, since they can be distributed almost anywhere. If we need only one, it is easier to name, and we can keep the pointers in hardware. Need ways to report full/empty status on a request.

4.11.2.2 Global locks

It may (or may not) prove useful to create locks which ensure globally that no more than one process has access to a data structure. Perhaps the general solution is a class of atomic read-modify-write functions.

4.11.3 Microcode

The microcoded engine is convenient for a couple of reasons: it has special-purpose functions, it is multi-threaded, and it has a fat pipe to memory. It’s inconvenient that it is hard to program (no compiler), has no cache, and behaves differently than other bus clients, besides the fact that it needs a separate design. I don’t think the special functions, aside from the fat pipe, are worth much. We could have had a multithreaded MIPS core with prefetch and flush instructions, and done more with less.

4.11.3.1 Buffer addressing

We need indexed addressing into the packet and (especially) memory buffers, so that we don’t have to dispatch to separate microinstructions to access the appropriate doubleword.

4.11.3.2 Buffer reset

It would be good to be able to clear the memory buffer in a single operation, to prevent leaks of information between processes.

4.11.4 Copy port

It was a mistake to try to short-circuit the fabric for local transfers. The local ports should have been a copy of the remote ports so that the hardware and microcode were exactly the same. Additional ports into this pile of latches should not be a problem.

4.11.5 Receive ports

The payload length needs to be writable for cases like deferred commands, where we want to combine the payload with a new header before writing to memory.

4.11.6 Cache

The DMA should have a cached, coherent interface to memory. The lack of coherent synchronization is (I suspect) going to prove to be a stumbling block.

4.12 Microcode

Chapter 5

DMA Engine

by Jud Leonard and Bryce Denney
[Last Modified $Id: DmaImpl.lyx 46805 2007-10-30 21:33:40Z denney $]

5.0.1 Package Attributes

Package

chip_dma_spec

Attributes

-dwaccessors

5.1 Introduction

The DMA Engine provides a high-bandwidth interface between the memory system and the fabric switch, relieving software of the low-level work of repetitively creating packets of memory data and injecting them into the fabric, or accepting packets from the fabric and distributing their payload to appropriate locations in memory.

This chapter describes the hardware of the DMA Engine. DMA Engine functions implemented by microcode,including the application-level software interface, are defined in another chapter.

5.2 Implementation

The ICE9 DMA engine is implemented as a programmable microengine that manages a set of TX and RX ports and an interface to the L2 cache. The microengine decides how to send outgoing packets and what to do with incoming packets, but relies on the other blocks to do nearly all data copying. Each of the TX and RX ports contain packet buffers, state machines, and address sequencers so that they can transfer to/from the fabric switch without consuming microengine cycles. The microengine reads its microcode from an instruction memory, which is initialized by system software at boot time. In each cycle it can perform an arithmetic operation on two 64-bit operands (A and B), producing a 64-bit result and a set of condition codes which can compute a branch target. Operands A and B generally read from the DMA's dedicated data memory (DmaDmem) but can also address registers in the TX and RX ports, and the cache interface.

Data moving through the DMA engine is stored in packet buffers while the DMA engine decides what to do with the packet and moves the data to the appropriate place. Imagine a packet that enters the chip on receive port 1 destined for this node. The packet arrives on receive port 1 of the fabric link logic, passes through the switch to the DMA, and is stored in the block labeled "RX Port 1" until the DMA engine processes the packet. Each RX port can hold up to four such packets at a time (approx 80x64 bits) before it must use backpressure to prevent the switch from sending any more data. As packets arrive from the switch, the RX port wakes up the appropriate thread in the DMA microengine by asserting rxpX_ue_BufAvail so that the microengine can examine the packet and take appropriate action. Usually the microengine will decide to copy the packet to main memory at a particular

address, and start a block transfer. The cache interface and receive port implement the block transfer and free up the packet buffer without any further interaction with the microengine.

Data moving in the other direction, from this node to the fabric, travel through the transmit ports in a similar way. Packets are transferred from main memory to a particular transmit port, e.g. TX Port 2 if the packet is destined for transmit port 2 onto the fabric. Each TX port can hold up to four such packets at a time (approx. 80x64 bits). When the transmit port raises txpX_ue_BufAvail, the microengine has a chance to decide how each packet should be handled. When the microengine is done, the transmit port sends packets out to the switch and recycles the packet buffer.

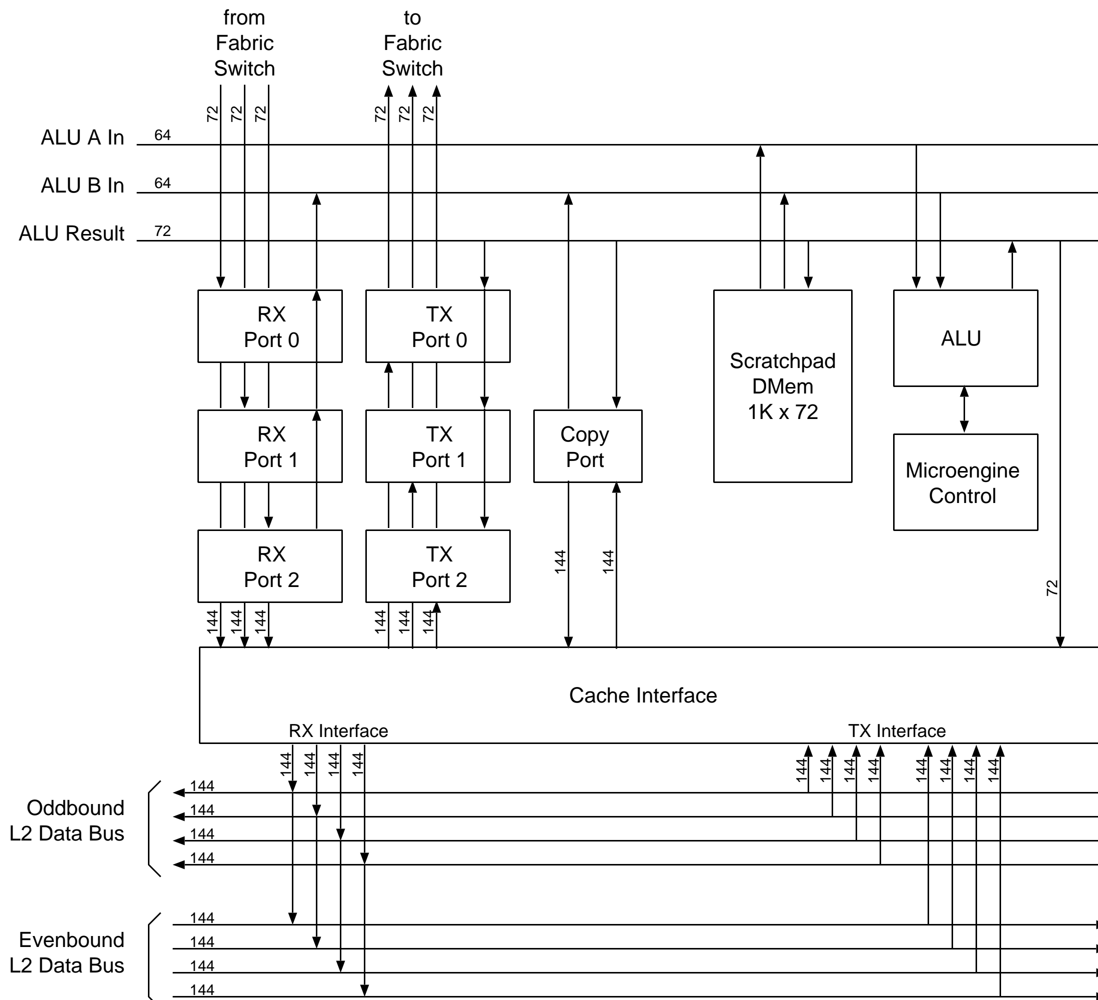
One other port, called the Copy Port, is used to send packets from one application to another within the chip. The copy port is designed to act very much like a transmit or receive port, so that hardware structures can be reused and library software can treat local (within the chip) and remote packet transfers in a similar way. The copy port can be used to perform traditional DMA memory-to-memory copies.

The microengine threads need to read and write L2 memory to manipulate queues and other data structures in memory. For this purpose, each microengine thread has a dedicated memory read buffer of 16 doublewords and a memory write buffer of 16 doublewords. The thread can schedule memory transfers into these buffers, wait until the transfer is complete, and manipulate the data. These buffers live in the Copy Port.

To service the ports, the microengine has about ten concurrent threads which contend for resources when they have something to do. Most threads are associated with a switch port (or the copy "port", or the queue manager). In addition, there is what might be thought of as a runt thread which has no preserved state, but which executes microinstructions to access datapath registers whenever an I/O reference to the DMA engine needs service.

5.2.1 Top Level Block Diagram

Here is block diagram of the DMA engine that shows the major blocks and data buses.



5.2.2 External Interfaces

5.2.2.1 Fabric Switch to DMA receive port X (X=0,1,2)

For each of the chip's three RX links, the fabric switch forwards data to three corresponding RX ports in the DMA engine. The interface for data traveling from fabric switch to DMA is described below. When no packets are being received, the data wires can be used for the fabric switch to send status information.

From	To	Signal	Description
dma	fsw	RdyX_s1a	DMA is ready to accept another packet (X=port 0,1,2). When the FSW begins a packet that consumes the DMA's last buffer, DMA deasserts dma_fsw_RdyX_s1a one cycle after the SoP.
fsw	dma	OutDatX_s2a<71:0>	data+ECC from fabric switch to DMA
fsw	dma	DatValX_s2a	DatVal_s2a contains valid data (X=port 0,1,2)
fsw	dma	SoPX_s2a	asserted when DatValX_s2a contains the header FORD
fsw	dma	EoPX_s2a	asserted when DatValX_s2a contains the trailer FORD

5.2.2.2 DMA transmit port X to Fabric Switch (X=0,1,2)

The DMA engine has three transmit ports corresponding to the chip's three transmit links. Each transmit port carries data to the fabric switch, which sends it to the appropriate link, using the interface described below. When there are no packets to transmit, the DMA can update the fabric switch's control registers.

From	To	Signal	Description
fsw	dma	BufAvailX_s3a	FSW is ready to accept another packet (X=port 0,1,2). The DMA samples fsw_dma_BufAvailX_s3a each cycle in order to decide whether it can begin a packet in the next cycle.
dma	fsw	InDatX_s0a<71:0>	data+ECC from fabric switch to DMA
dma	fsw	DatValX_s0a	DatVal_s2a contains valid data
dma	fsw	SoPX_s0a	asserted when DatValX_s2a contains the header FORD
dma	fsw	EoPX_s0a	asserted when DatValX_s2a contains the trailer FORD

5.2.2.3 DMA to L2 Cache Switch

See 7.2 in L2 Cache chapter.

The DMA can start one CmdAddr transaction per cycle and one Data transaction per cycle onto the L2 cache switch buses. In each cycle it may request the even CmdAddr bus or the odd CmdAddr bus, but never both directions at once. Also, it may request the even Data bus or the odd Data bus, but never both directions at once. Meanwhile, the DMA can accept one incoming CmdAddr transaction and one Data transaction per cycle.

The DMA engine can have up to four outstanding block reads and four outstanding block writes to the L2 cache. In addition it responds to I/O reads and writes from the six processors.

5.2.3 Module Hierarchy

Before diving into the details of each component, here is a tree that shows how the DMA engine is organized into modules and submodules.

- Dma: top level of DMA engine
  - DmeUe: microengine control logic
    - RAM containing microengine instructions
  - DmaAlu: microengine ALU
  - DmaDmem: microengine data memory
  - DmaCif: L2 cache interface
    - several queues to keep track of outstanding requests
  - DmaRxp: contains the three RX ports that receive from fabric switch

    * DmaRxpCtl0: RX port logic for port 0
    * DmaRxpCtl1: RX port logic for port 1
    * DmaRxpCtl2: RX port logic for port 2
    * packet buffers

  – DmaTxp: contains the three TX ports that transmit to fabric switch

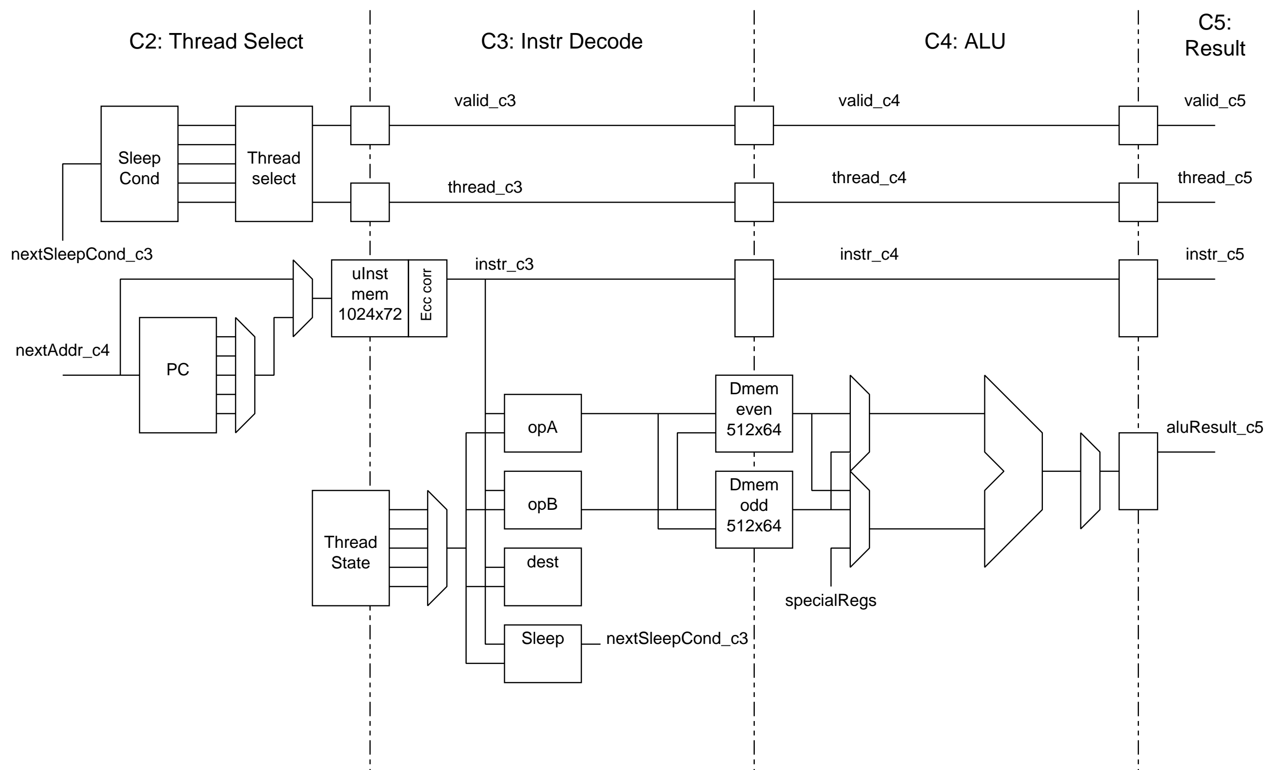
    * DmaTxpCtl0: TX port logic for port 0
    * DmaTxpCtl1: TX port logic for port 1
    * DmaTxpCtl2: TX port logic for port 2
    * packet buffers

  – DmaCopy: copy port, for memory-to-memory transfers

    * packet buffers
    * memory read buffers
    * memory write buffers

5.2.4 DmaUe: Microengine Control Logic

The microengine is implemented with a four-stage pipeline, consisting of thread selection (C2), instruction decode (C3), ALU (C4), and write result (C5).



The following table describes the pipeline for the microengine in more detail.

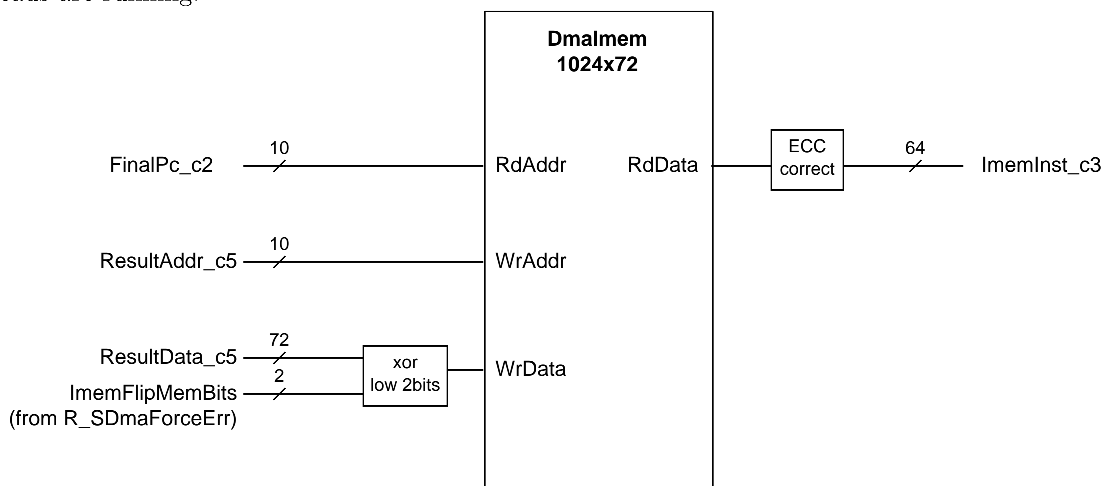
Stage	Name	Description
C2	Thread Sel	Choose thread to run next using round robin scheme. Once a thread runs, it must wait a cycle before it can run again. Find the program counter for the selected thread. If the same thread is selected in C4, the next uPC is bypassed from C4 instead. The bypass allows us to avoid having a branch delay slot.
C3	Instruction Decode	Read microinstruction memory on rising edge. Decode instruction. Prepare to read operand memories/registers by driving OpaAddr and OpbAddr.
C4	ALU	All operands are read from respective memories on the rising edge of C4 and sent to the ALU. The ALU result is computed and registered at then end of C4. Compute the NextAddr for the thread, and bypass it back to C2 in case the same thread is selected again. Prepare to write results to memory by driving the ResultAddr and ResultData buses.
C5	Write Result	Write ALU result to selected memory on rising edge. Write changes to thread state registers. If necessary, ask the cache interface to start a memory transfer using the TaskStart interface.

FIXME: Document how I/O reads and writes get into the microengine and how it deals with them.

5.2.5 DmaImem: Microengine Instruction Memory

The DmaImem contains microinstructions that the DMA engine will execute. The instruction memory is initialized using WTIOs from a processor, while the DMA microengine is idle (all threads disabled). The data to be written flows through the datapath and ends up on alu_xxx_ResultDat_c5a<71:0>, which contains both data and ECC. If ImemFlipMemBits are used, the data can be intentionally corrupted before being written.

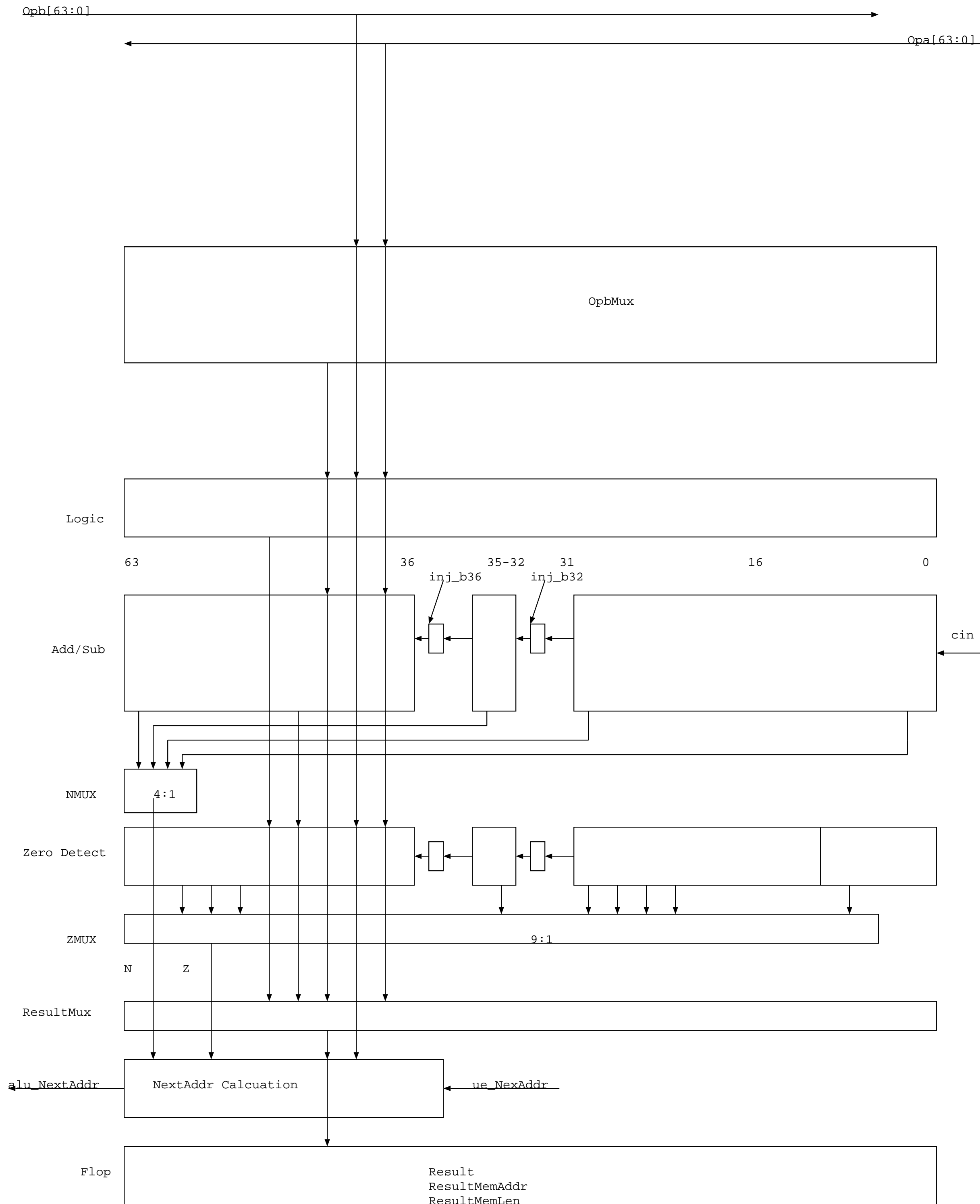
Once the DMA threads are enabled, the microengine reads one instruction of DmaImem per cycle, does ECC correction, decodes the instruction, and executes it. It is unsafe for a processor to write Imem while the DMA threads are running.



Implementation note: For timing reasons, the Imem is implemented as four banks of 256x72, interleaved on bits 1 and 0 of the read address. In each read cycle, all four banks are read in late C2, and the correct result is selected based on address bits 1:0, which arrive from the ALU in early C3.

5.2.6 DmaAlu: Microengine ALU

The microengine ALU is designed to calculate memory addresses and queue pointers. It also contains some general arithmetic such as add and subtract, booleans, etc.



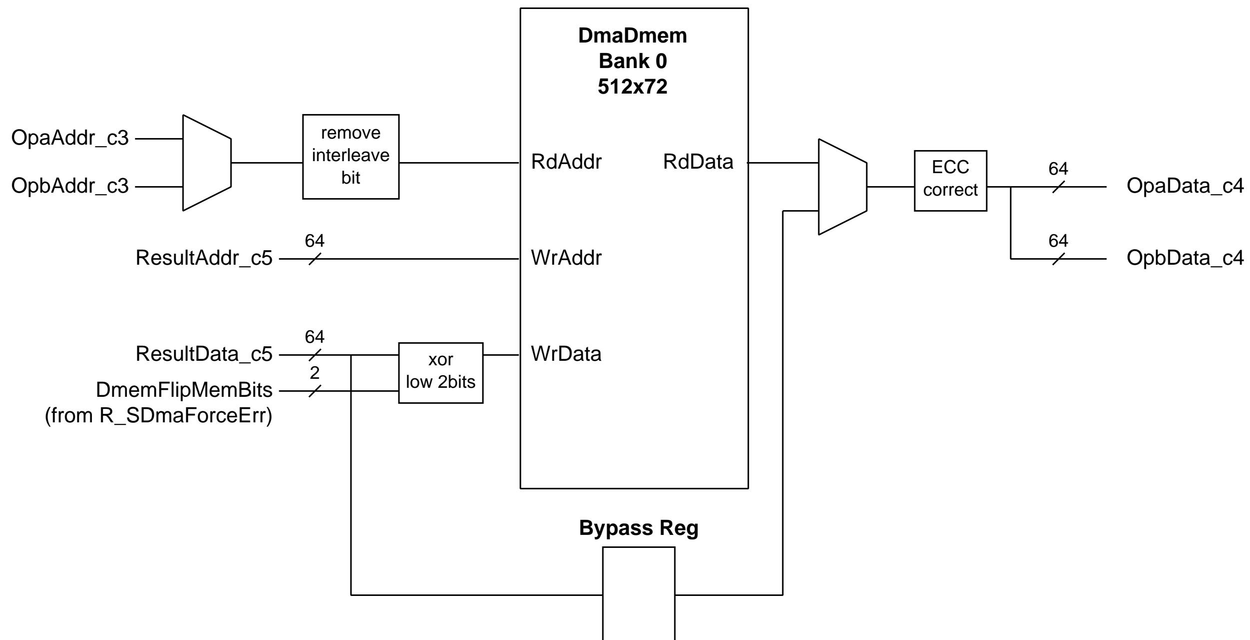
5.2.7 DmaDmem: Microengine Data Memory

The Dmem is the microengine's scratchpad memory. It can be read and written by every instruction, and is also accessible to processors via I/O reads and writes. The Dmem is divided into four banks of 256 words by 64 bits each, and operands A and B can address the banks independently. Operands A and B can read different addresses

from the four banks of DmaDmem. However, if the two operands try to access different addresses in the same bank in the same instruction, operation is undefined. The hardware simulation models will provide asserts to detect this condition.

Since a thread may execute every two cycles, a potential data hazard exists between results written in C5 and operands read in C3 from the same address. The register file does not like to be written and read at the same address. To avoid the hazard, a bypass register allows ResultDat_c5 to be delayed until C6 and then driven onto the operand A or B data bus when the read and write addresses match.

One bank of the DmaDmem is described in the diagram below. The Dmem is interleaved on bit numbers (DMEM_INTERLEAVE_BIT and DMEM_INTERLEAVE_BIT+1), presently 4 and 5. To produce addresses for a given Dmem bank, the interleave bits must be removed.



Dmem Address Bits	Assignment
Address<9:8>	00->Process Variables
Address<7:4>	Variable selection
Address<3:0>	Process Index (0-13)
Address<9:8>	01->Thread Variables
Address<7:4>	Variable selection
Address<3:0>	Thread number (0-9)
Address<9>	1->Context Variables
Address<8:6,4>	Variable Selection
Address<5>	1->Transmit, 0-> Receive
Address<3:2>	Port number
Address<1:0>	Context index

We have allocated a 1K x 64 register file to hold control/status information (12 processes * 16 doublewords), (10 threads * 16 doublewords), and contexts (2 directions * 4 ports * 4 contexts * 16 doublewords).

Each of the transmit and receive threads (including the “copy” instances of each) has four hardware contexts for which it is responsible; each such context consists of 16 doublewords which can be used as needed by microcode. The allocation is chosen to correspond closely with the structure of commands, so that a queue entry can be loaded directly into the context memory.

The current assignment for transmit contexts is:

Figure 5.1: DMem Process Variables (0-255)

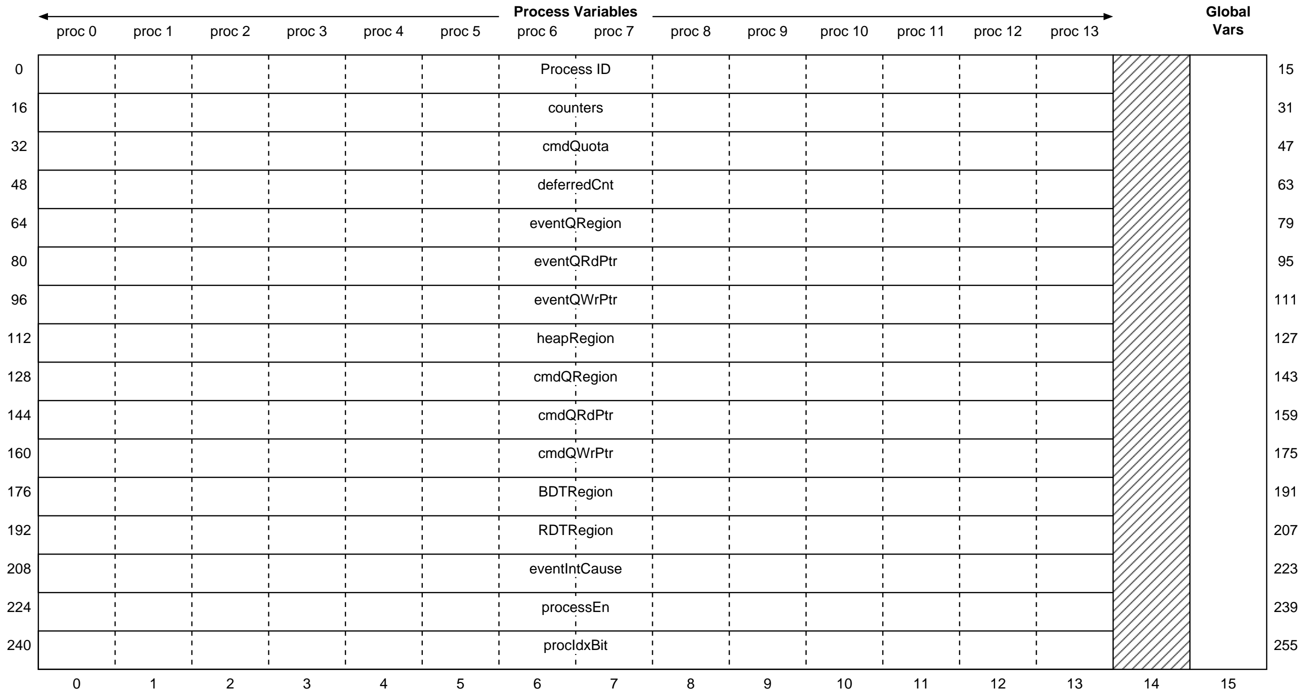


Figure 5.2: DMem Thread Variables (256-511)

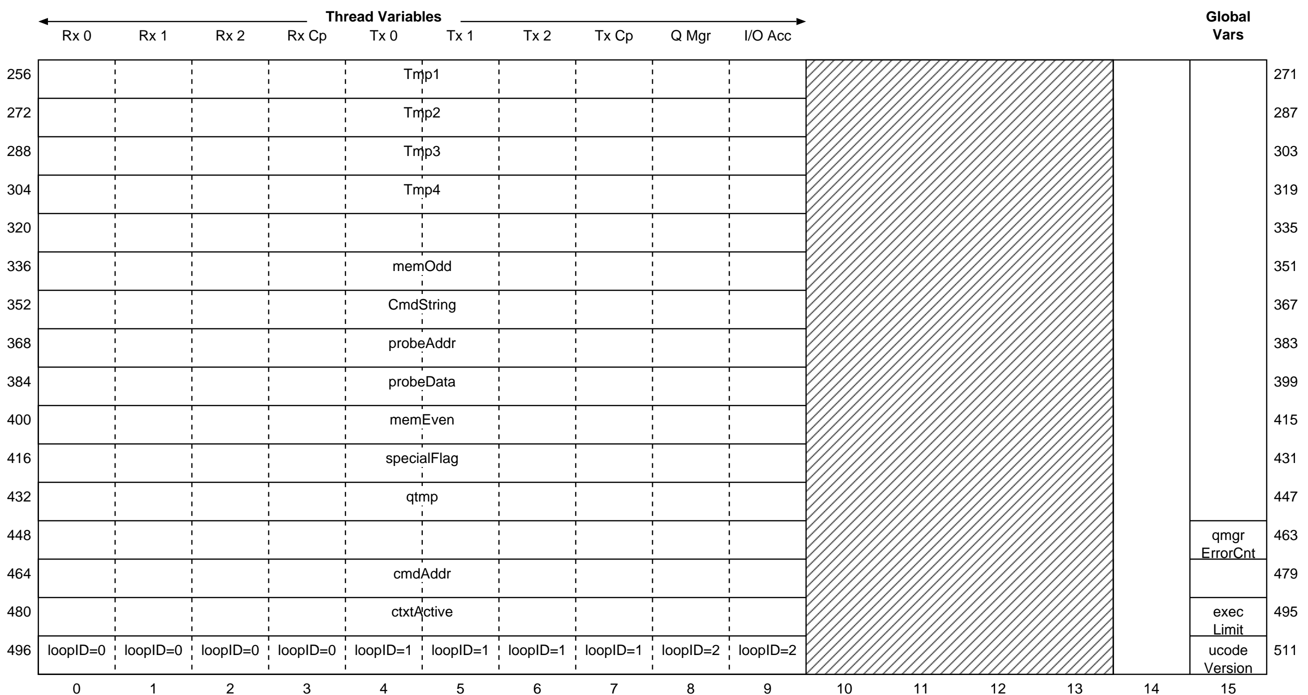


Figure 5.3: DMem Tx Context Variables (512-767)

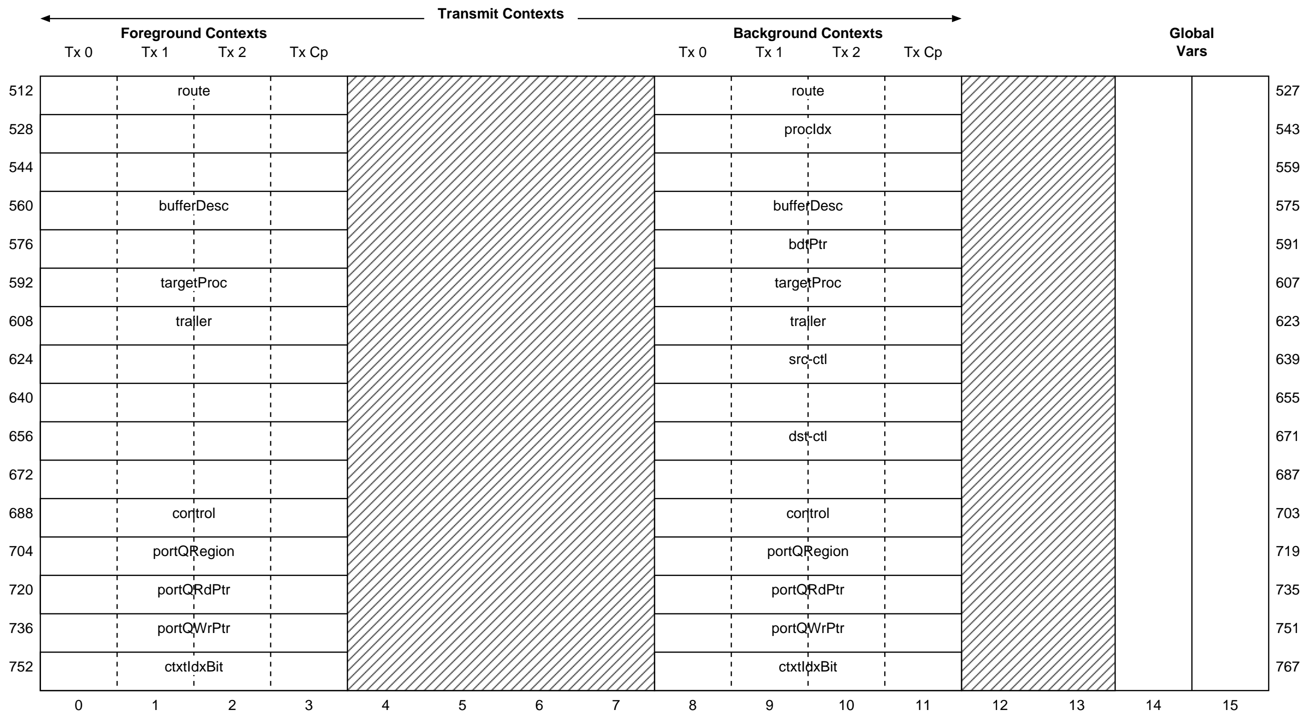
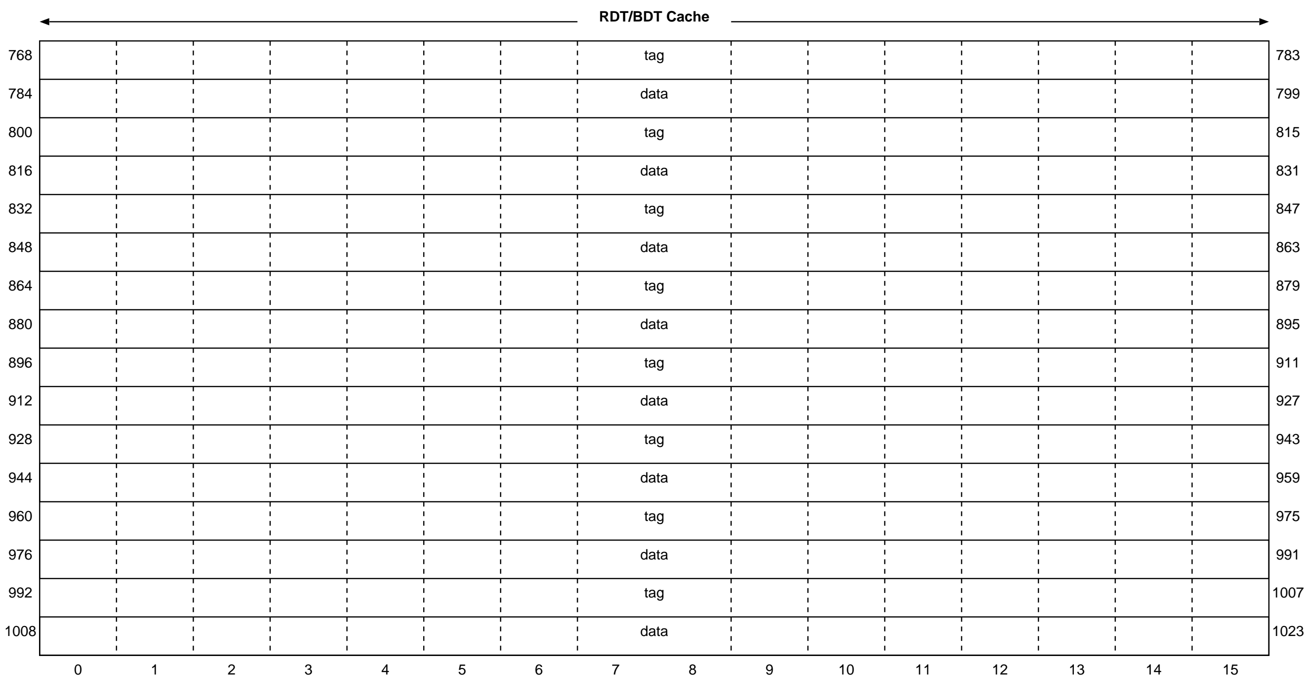


Figure 5.4: DMem Rdt/Bdt Cache (768-1023)



Dword	Bits	Function	Description
0	7:0	command	DMA command to perform
0	63:8	header	RDT data to be put in header ford
1	63:32	context info	context id, remote context index, local process index
1	31:0	segment len	remaining segment length
2	63:0	buffer descr	BDT data combined with offset
3	31:0	buffer handle	BDT index
4-7			Unused

For receive contexts, the assignment is:

Dword	Bits	Function	Description
0	63:0		Unused
1	63:32	context info	context id, local process index
1	31:0	segment len	remaining segment length
2	63:0	buffer descr	BDT data combined with offset
3	31:0	buffer handle	BDT index
4	7:0	notifier cmd	Command to queue upon completion
4	63:8	notifier head	RDT data for notifier header
5	63:0	notifier ctl	Control word of notifier, filled by microcode
6-7	63:0	notifier pay	Software payload for notifier message

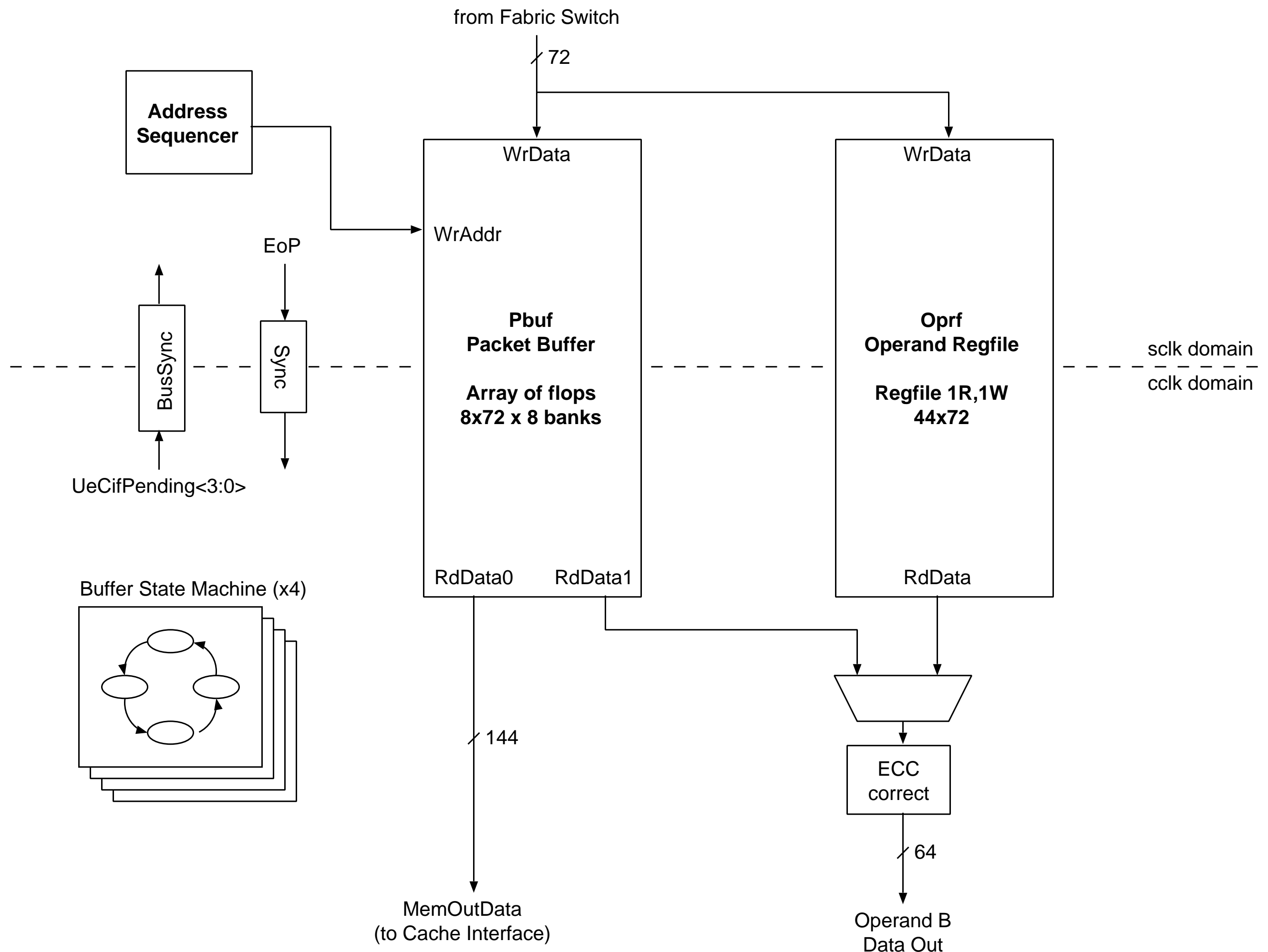
5.2.8 DmaRxp: Receive Ports

The DmaRxp module contains three instances of the receive port, connected to the three data ports coming from the fabric switch.

A DMA receive port queues packets as they come from the fabric switch, one doubleword (64 bits) per cycle. The header, control, and trailer FORDs are captured into one register file (Oprf), while other doublewords are stored in a packet buffer (Pbuf) which can be quickly dumped to memory through the cache interface, eight doublewords (512 bits) per cycle. The DMA microengine decides what should be done with the packet: either throw it away or schedule it to be transferred to main memory. The Pbuf can store DMA_PBUF_N different packets (presently 4) before it must tell the switch to hold off until another buffer is available. Both Pbuf and Oprf are readable on operand B at a rate of 64 bits per cycle.

CAUTION: The receive port control logic uses uncorrected data from the fabric switch in several cases. The uncorrected HasCtrl bit in the header is used to determine whether the second Ford is to be treated as a control Ford or payload. The uncorrected ProcessIndex in the EoP is sampled into registers, which are retimed into the cclk domain and drive the rxpN_ue_ProcessIndex ports. In February 2006, we decided that using the uncorrected data was acceptable because of the way the fabric switch drives data to DMA. The FswDmao block corrects and generates new ECC just before driving 72 bits of data to the DMA engine, so the DMA will always see good ECC coming from the FSW (barring logic or interconnect problems of course). Because the ECC is known to be correct, we will continue to use uncorrected bits for the purposes of HasCtrl and ProcessIndex only. The HLM contains assertions that complain if these bits are ever corrupted (by doing an ECC correction and checking which bit was flipped) so that we will know if this condition ever occurs. See Bug1143 and Bug1160.

Dma Receive Port Block Diagram



The Pbuf is organized as DMA_PBUF_N different buffers of DMA_PBUF_WORDS words. For DMA_PBUF_N=4, the address into the register file looks like:

bits 6:2	bits 1:0
offset	buffer number

The maximum offset is not a power of two, but it's easy to make the maximum buffer number a power of two. We chose DMA_PBUF_N = 4. By putting the buffer number in the low order bits, we can populate as many offsets as we wish without wasting memory. If DMA_PBUF_N=4 and DMA_PBUF_WORDS=19, the memory size is 19*4 = 76 words. The Pbuf must be implemented in a way that supports 128-bit reads of offset N and offset (N+1) for even N.

The Oprf is used for two purposes: several words are used to store the header ford, control word, trailer word, etc. for each packet. In another part of the Oprf, we store status information from the fabric switch. The status information can be read through the operand bus so that software running in the cores can access it. The physical organization of Oprf is:

Oprf address	Description
0x00 - 0x1F	Switch status information
0x20 - 0x2F	RX port control registers for each buffer

For addresses 0x20-0x2F, the Oprf address decodes as follows

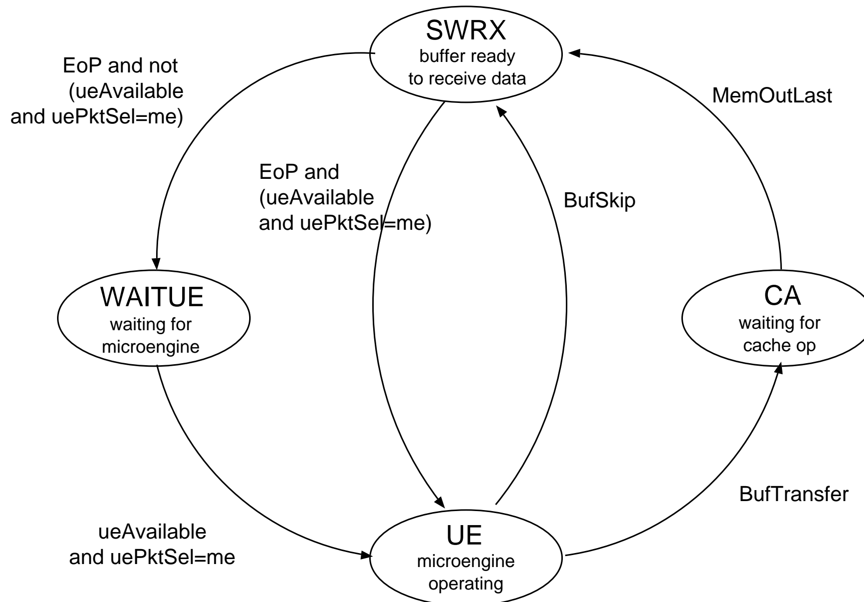
bit 5	bit 4	bits 3:2	bits 1:0
1	0	register num	buffer num

Reg number definitions for Oprf bits 3:2 are:

0	unused
1	header ford
2	control ford
3	trailer ford

The receive port contains a buffer state machine for each of the DMA_PBUF_N packets in the buffer. Each buffer state machine is independent of the others, except that only one buffer may be in state UE at a time. The state diagram for each buffer is shown in Figure ??.

Dma Receive Port Buffer States



The DmaRxp module spans two clock domains, sclk (switch clock) and cclk (core clock). The data arrives in sclk time, and is written into the Pbuf on sclk edges. When a packet is completely transferred, the microengine and cache controller (running on cclk) read the data when it is known to be stable. A 4-state FSM per buffer keeps track of which buffer is being used in which way. The register file is the primary means of synchronizing data across domains, but several control signals need to pass across clock domains using synchronizers.

EoP (sclk to cclk) is produced by the switch when it sends the last doubleword in a packet. When EoP comes from the switch, it is a one-cycle pulse in sclk. This passes through a pulse synchronizer[1] and becomes a one-cycle pulse in cclk. In the cclk domain, EoP tells the state machine that a buffer is completely transferred and ready to be used by the microengine. EoP causes the state transition from ST_SWRX to either ST_WAITUE or ST_UE.

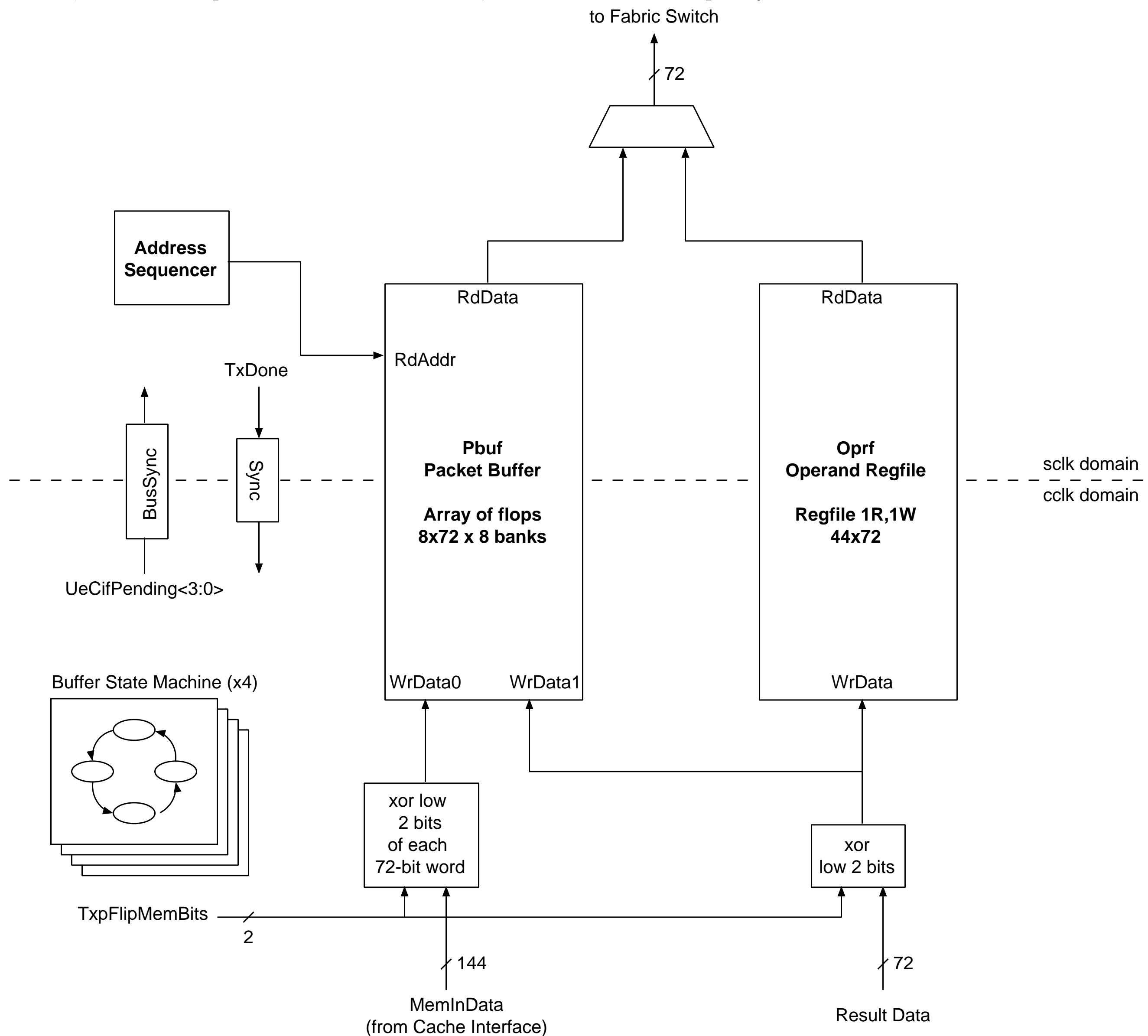
UeCifPending<3:0> (cclk to sclk) is a bit vector produced in the cclk domain that tells the sclk logic whether a buffer is in use (by microengine or cache interface) or ready to receive another packet. Individual bits of UeCifPending are set by the arrival of EoP and cleared when the microengine and cache interface are done with the packet. To address the dangers of sending a 4-bit bus through separate synchronizers, the bits are sent using a module BusSyncOneWay which implements a handshake protocol and resamples into the destination clock domain in a safe way.

5.2.9 DmaTxp: Transmit Ports

The transmit ports are similar to the receive ports except that the data flows from the L2 cache to the DMA transmit port to the fabric switch. The microengine can either ask the cache interface to write a packet's payload into a packet buffer, or write it directly via the Result data bus. The microengine also writes the header, control,

[1] A one-cycle pulse in the source clock domain generates a toggle signal which is passed through a synchronizer. When any transition is detected in the destination clock domain, it is turned into a one-cycle pulse in the destination clock domain.

trailer FORDs, and the payload length into registers in the Oprf. Then the address sequencer in the transmit port takes over, and sends a packet to the fabric switch, 64 data + 8 ecc bits per cycle.



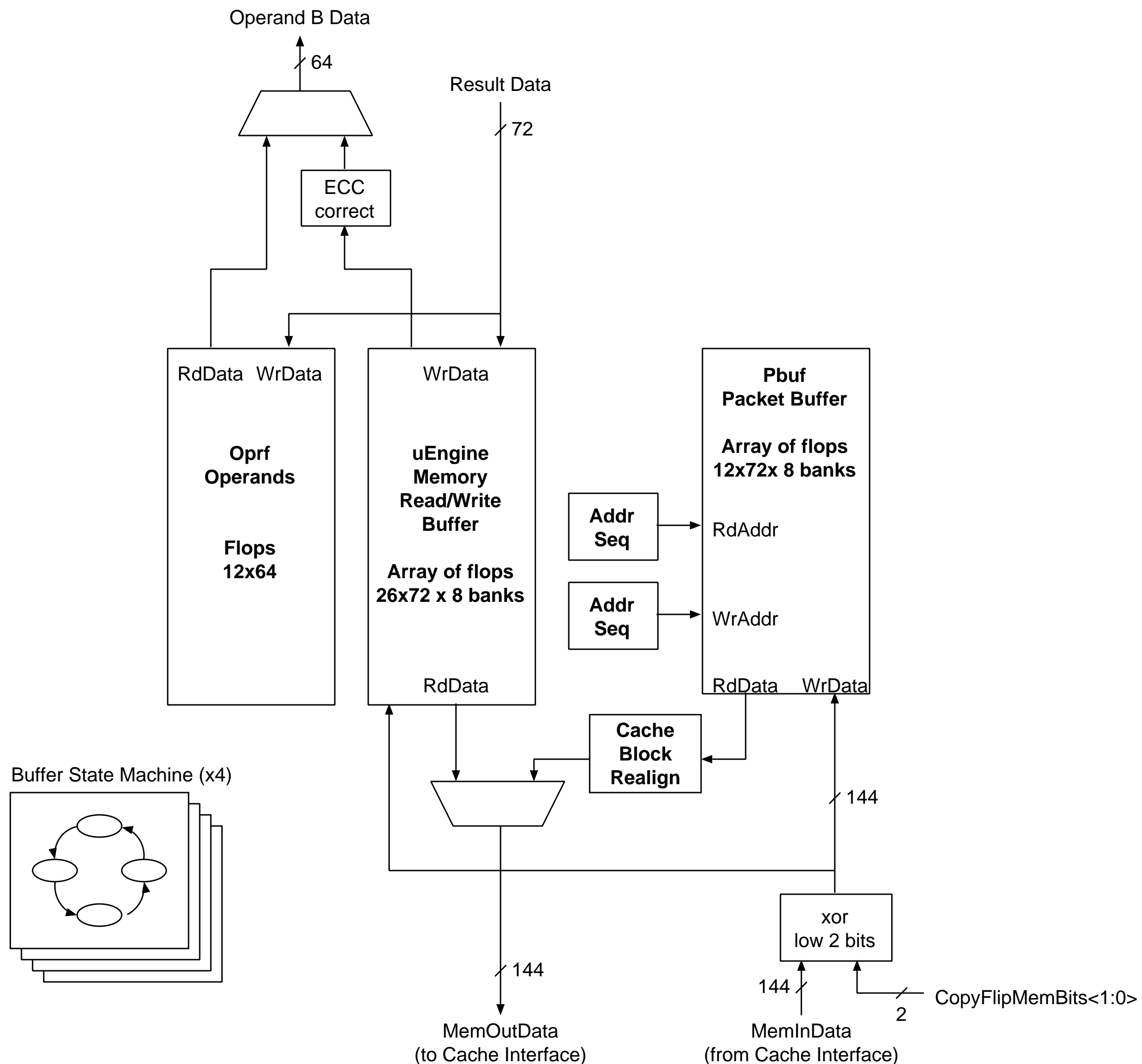
Unlike the receive port, the transmit port may need to read packets from memory which are not aligned in a convenient way. The packet payload may start on any 32-bit boundary (memory address is a multiple of 4). To handle unaligned packets, the Pbuf is large enough to hold 3 cache blocks per packet, and a 32-bit alignment mux is placed on the path to the fabric switch. If the packet payload is aligned, only two cache blocks are needed and the data is driven to the fabric switch starting at address 0. But in unaligned cases, the DMA cache interface may need to read three cache blocks into the Pbuf, knowing that some bits will not be used, and then read out just the relevant data.

5.2.10 DmaCopy: Copy Port

The copy port is used when sending packets to destinations within the node. Packets are loaded into into a packet buffer from the source address and then written back to memory at the destination address. The microengine treats the copy port as a transmit port and a receive port, and in fact the transmit and receive functions of the copy port are managed by separate threads.

Like the transmit port, the copy port needs to be able to read packet payloads at various alignments, down to any 32-bit boundary. Data is written to the Pbuf aligned exactly as it is in main memory, then realigned properly by the cache block realignment module as it is read out.

The copy port also contains a memory read/write buffer (RWMB), which give the microengine a way to read and write cache lines to/from the memory system directly. The RWMB is 10 threads * 16 doublewords = 160 words by 64 bits, plus 6 extra words to assist I/O operations to/from the 6 processors.



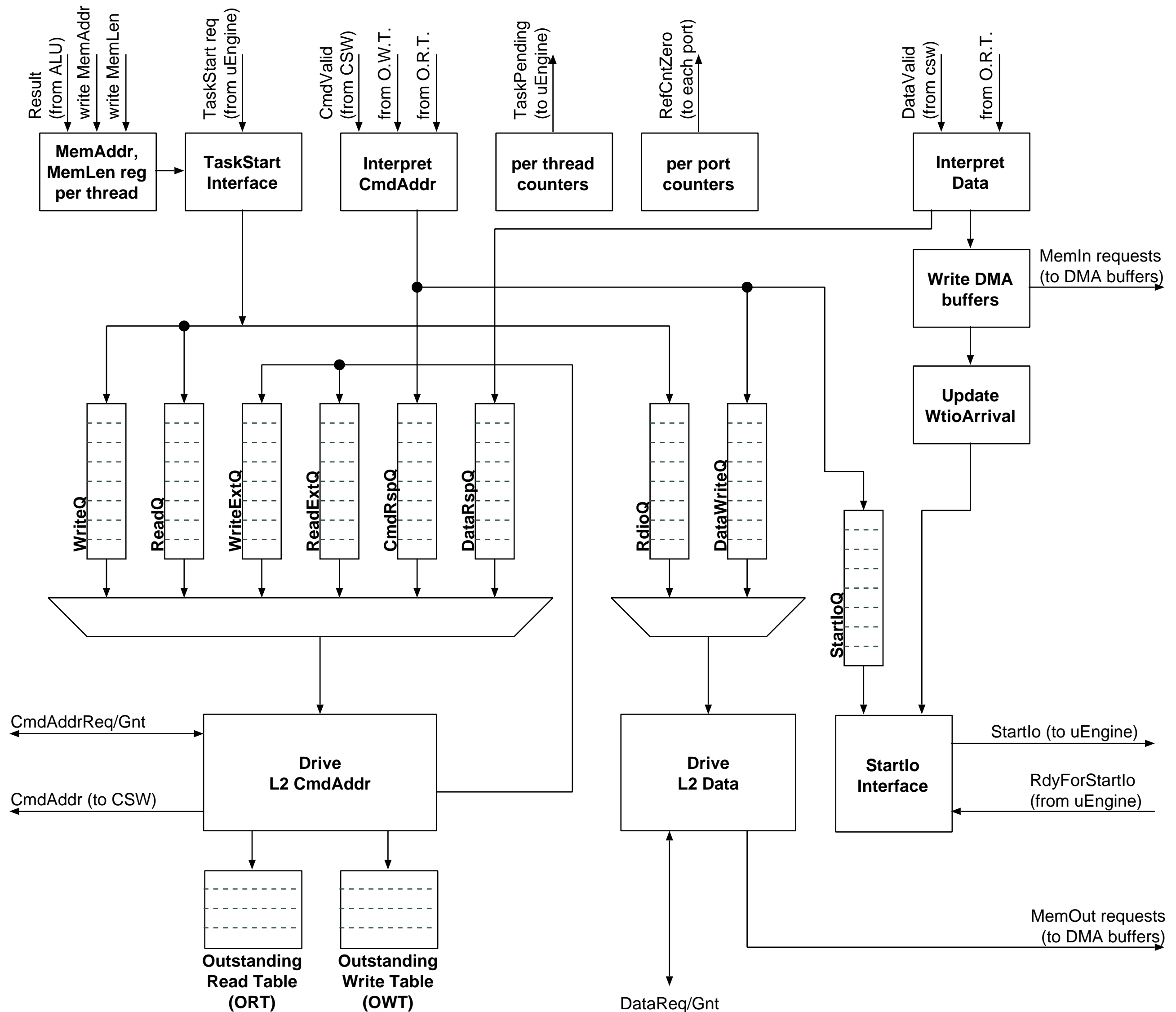
5.2.11 DmaCif: Cache Interface

The cache interface manages transfers between the L2 Cache Memory Bus and buffers inside the DMA block. The DmaCif handles the details of the L2 memory bus protocol: requesting the CmdAddr bus and the Data buses and handling I/O reads and writes from the processors. Each microengine thread can start memory transfers or "tasks" via the TaskStart interface and optionally wait for its memory transfers to complete. The TaskStart interface determines the memory address and length of transfer by copying the MemAddr and MemLen register value for the requesting thread. (Exception: some transfers specify a fixed length so the MemLen value is not always used). Tasks are placed in queues where they wait for their turn to use the CmdAddr or Data bus. Memory transfers move data between main memory and the TX, RX, and Copy port buffers in the DMA engine by driving the MemIn and MemOut interfaces. MemIn controls data moving from main memory into DMA buffers. MemOut controls data moving from the DMA buffers out to main memory. As I/O reads and writes arrive from the CSW, they are sent to the microengine via the StartIo interface.

The per-thread counters and per-port counters keep track of how many requests are waiting in queues or outstanding read/write tables, so that the DmaCif can notify the interested parties when the requests that affect them are finished. As a task is first processed in TaskStart, the per-thread and per-port counter is incremented. In

the queues, each task is tagged with the microengine thread number that initiated it, so that the correct counter can be decremented when the task completes. The outputs of the thread and port counters are TaskPending and RefCntZero. TaskPending<9:0> tells the microengine which threads have outstanding memory requests. One RefCntZero signal goes from the cache interface to each port (rx, tx, copy) telling it whether there are any outstanding memory requests.

A block diagram of the DmaCif is below.



For each type of traffic that passes through the DmaCif, the next few paragraphs will describe what path the requests follow. The traffic types are

Symbol	Name
BRD	block read
BWT	block write
RDIO	I/O read
WTIO	I/O write
SPCL	special
INTR	interrupt

Block Read

The microengine drives the TaskStart interface, and the request is placed in ReadWriteQ. The request cannot leave ReadWriteQ until an ORT entry is available; the number of the selected ORT entry (0-3) determines which of the DMA's transaction IDs will be used. When the request comes out of the queue, the DmaCif arbitrates for the CmdAddr bus in the appropriate direction and drives a BRD command onto the bus. The ORT entry is written

with the details of this block read request, so that we know how to handle the data when it arrives. If more cache blocks are required to finish the request, the next cache block request is written into ReadWriteExtQ. Wait for data to be returned or for a PRBNOHIT.

If a PRBNOHIT is seen on the CmdAddr bus, the Interpret CmdAddr block looks up the entry in the ORT to find the address, then places a BRDR request into the BrdrQ.

When data arrives from the CSW, it enters the Interpret Data block, which uses the TID to find the corresponding ORT entry. From the ORT we know which of the DMA buffers will be written and the starting address. The DmaCif drives the MemIn interface to tell the DMA buffer to write, and data from CSW flows into the buffer. The Interpret Data block also places a PRBDONE request into the DataRspQ if needed (only if DataOrigin indicates that the data did not come from a coherence controller.)

When a microengine thread starts a read operation, the cache interface to increment a per-thread counter by one. When the transfer is finished (the data is ready to be used by microcode), the counter decrements by one. If the thread decides to sleep until memory operations are done, this per-thread counter controls when the thread wakes up.

Block Write

The microengine drives the TaskStart interface, and the request is placed in ReadWriteQ. The request cannot leave ReadWriteQ until an OWT entry is available; the number of the selected OWT entry (0-3) determines which of the DMA's transaction IDs will be used. When the request comes out of the queue, the DmaCif arbitrates for the CmdAddr bus in the appropriate direction and drives a BWT command onto the bus. The OWT entry is written with the details of this block write request, so that we know what to do when the "go" command arrives. If more cache blocks are required to finish the request, the next cache block request is written into WriteExtQ. Wait for a BWTGO, BWTNOHIT, or PRBINV command.

When the BWTGO, BWTNOHIT, or PRBINV command arrives from the CSW, it enters the Interpret Command block, which uses the TID to find the corresponding OWT entry. From the OWT we know which of the DMA buffers will be sent to memory and the starting address. The DmaCif drives the MemOut interface to tell the DMA buffer to send, and data from the buffer flows into the CSW. For BWTNOHIT or PRBINV, the data is sent to the even or odd coherence controller; for BWTGO the data is sent to the module that sent the BWTGO based on CmdAddrOrigin.

When a microengine thread starts a write operation, the cache interface to increment a per-thread counter by one. When the transfer is finished (the data has been sent to the CSW), the counter decrements by one. If the thread decides to sleep until memory operations are done, this per-thread counter controls when the thread wakes up.

I/O Read

A RDIO command arrives from the CSW and enters the Interpret CmdAddr block. The request is placed in the StartIoQ where it waits to enter the StartIo interface. Eventually it reaches the head of queue and is driven to the microengine. The microengine completes the I/O read operation and puts the result into a known location in the Copy Port's Write Memory Buffer. Then the microengine drives the TaskStart interface to ask DmaCif to respond to the I/O read. The request is placed in the DataRdioQ, we arbitrate for the Data bus and drive MemOut to read the data from the copy port. Finally the data moves from the copy port to the CSW to complete the I/O read operation.

The per-thread counters are not affected by I/O reads.

NOTE: The DMA contains bug1991 in which RDIO can be corrupted by a WTIO following it from the same core. See 8 for details.

I/O Write

A WTIO command arrives from the CSW and enters the Interpret CmdAddr block. Three things happen: 1) an RDIO request is placed in CmdRdioQ, 2) the details of the WTIO command are placed in StartIoQ, and 3) a bit is cleared in WtioDataReady<5:0> a bitmask that records whether the write data has arrived or not. WtioDataReady is indexed by core number. The RDIO is not allowed to issue from the CmdRdioQ until the WTIO has reached the head of the StartIoQ. When the WTIO reaches the head of the StartIoQ, the RDIO goes out onto CmdAddr to the processor. Then we must wait for the core to send data.

When the data arrives, it enters the Interpret Data block, which uses the TID to know which core sent the data. Knowing the core number, we know where the write data is supposed to go. The DmaCif sends a MemIn request to the copy port to put the I/O write data into the Memory Read Buffer in the copy port. The Interpret Data block also sets WtioDataReady<corenum> so that it is allowed to issue from the StartIoQ.

Finally, the WTIO request issues from the StartIoQ and is sent to the microengine. The microengine completes the I/O write operation by reading from the copy port and writing the data into the memory selected by the address

from the CmdAddr cycle.

The per-thread counters are not affected by I/O writes.

NOTE: The DMA contains bug1991 in which RDIO can be corrupted by a WTIO following it from the same core. See 8 for details.

SPCL (Special) Command

A SPCL command is treated like an I/O Read because it is triggered by just a CmdAddr cycle. The SPCL arrives from the CSW and enters the Interpret CmdAddr block. The request is placed in the StartIoQ where it waits to enter the StartIo interface. Eventually it reaches the head of queue and is driven to the microengine. The microengine completes the SPCL operation, then drives the TaskStart interface to ask DmaCif to respond to the SPCL. The request is placed in the SpclIntQ, we arbitrate for the CmdAddr bus and send the DONE command back to the core.

The per-thread counters are not affected by I/O writes.

Interrupts

Microcode causes an interrupt by setting the memOp field to “sendIntr” and placing 16 bits of interrupt data on the alu result. The alu result bits 15:12 are the bus stop number to deliver to, and alu result bits 11:0 are the unique number that tells the processor which interrupt fired. The INTR command is placed in the SpclIntQ. When it reaches the head of queue, we arbitrate for the CmdAddr bus and send the INTR command to the core. There is no response.

The INTR operation increments the DmaCif’s thread counter by one. When the interrupt has been sent on the CmdAddr bus, the thread counter decrements again.

5.2.11.1 Cache Interface Queues

For each of the queues in the block diagram, the table below tells the size, data representation, and what types of commands would use the queue.

Here is the table for ICE9:

Queue	Data type	Length	Commands	Notes
ReadWriteQ	DmaCifTask	20	BRD, BWT	10 threads * 2 reqs per thread
ReadWriteExtQ	DmaCifTask	20	BRD, BWT	10 threads * 2 reqs per thread
CmdRdioQ	DmaCifProtocolEntry	6	RDIO	6 cores
BrdrQ	DmaCifProtocolEntry	4	BRDR	4 outstanding reads
SpclIntQ	DmaCifProtocolEntry	16	SPCL DONE, INT	6 SPCL response + 10 INTRs
DataRspQ	DmaCifProtocolEntry	4	PRBDONE	4 outstanding writes
DataRdioQ	DmaCifProtocolEntry	6	RDIO	for 6 cores
DataWriteQ	DmaCifProtocolEntry	4	BWT	4 outstanding writes
StartIoQ	DmaCifStartIoEntry	12	WTIO,RDIO	6 cores * (1 read or SPCL + 1 write)

Twice9 the number of outstanding reads and writes changed, and the number of cores changed.

Queue	Data type	Length	Commands	Notes
ReadWriteQ	DmaCifTask	20	BRD, BWT	10 threads * 2 reqs per thread
ReadWriteExtQ	DmaCifTask	20	BRD, BWT	10 threads * 2 reqs per thread
CmdRdioQ	DmaCifProtocolEntry	10	RDIO	10 cores
BrdrQ	DmaCifProtocolEntry	7	BRDR	7 outstanding reads
SpclIntQ	DmaCifProtocolEntry	20	SPCL DONE, INT	10 SPCL response + 10 INTRs
DataRspQ	DmaCifProtocolEntry	7	PRBDONE	7 outstanding writes
DataRdioQ	DmaCifProtocolEntry	10	RDIO	for 10 cores
DataWriteQ	DmaCifProtocolEntry	7	BWT	7 outstanding writes
StartIoQ	DmaCifStartIoEntry	20	WTIO,RDIO	10 cores * (1 read or SPCL + 1 write)

5.2.11.2 Interfaces in DmaCif

Interface	Description
TaskStart	The microengine asks cache interface to start a data transfer using the TaskStart interface.
StartIo	The cache interface notifies the microengine that an I/O read or write has occurred. The microengine sends back a status signal that tells when another request can be sent.
MemOut	The MemOut bus carries data from packet buffers out of the DMA buffers to L2 memory. MemOut is connected to the three receive ports, the RX side of the copy port.
MemIn	The MemIn bus carries data from memory into the DMA packet buffers. MemIn is connected to the three transmit ports and the TX side of the copy port.
L2 Cache Interface	The DMA can arbitrate and write commands, then arbitrate and write data onto the L2 Cache memory bus. The CSW can carry data in either direction. When other blocks write to the DMA via the memory bus, the cache switch hands the DMA one CmdAddr value and one Data value per cycle.

5.2.11.3 TaskStart Interface (Microengine to DmaCif)

The microengine requests memory transfers using the TaskStart interface of the cache interface, described in Table 5.1. The timing of the TaskStart interface signals is described in Figure 5.5.

The cache interface contains four queues which record memory transaction requests. When the microengine requests a transfer (raises TaskStart), that thread's current MemAddr is placed into one of the queues along with all the parameters of the transfer. Read and write tasks are placed into separate queues, so that writes cannot get stuck behind reads and vice versa. Some memory transfers are several cache lines long and must be done in several steps. As a step is completed, if there is more to be transferred, a new task is placed at the tail of another queue, called the "extended" queue. So, the cache interface contains a total of 4 queues: WriteQueue, WriteExtendedQueue, ReadQueue, and ReadExtendedQueue. The cache interface will pick the operation at the head of one of the queues and work on it until completion, then pick another in the next cycle.

5.2.11.4 StartIo Interface (DmaCif to microengine)

Table 5.1: **TaskStart Interface from Microengine to Cache Interface**

Signal	From	To	Cycle	Description
TaskStart	Ue	Cif	C5	When asserted, causes a memory transfer to start
TaskThread<2:0>	Ue	Cif	C5	Tells which microengine thread has started a memory transfer. Each thread has its own MemAddr value, so TaskThread tells which value to use. Also, the cache interface keeps record of how many transfers are pending per thread and reports back to the microengine.
TaskTarget<5:0>	Ue	Cif	C5	Which of the DMA's memories will be accessed
TaskType<1:0>	Ue	Cif	C5	Is it a read or a write operation, and what kind? The types are cacheline read, cacheline write, I/O read response, and I/O write response. See 5.5.31 for encoding.
TaskTid<5:0>	Ue	Cif	C5	Transaction ID of the task. This is only valid for I/O operations.
TaskOrigin<5:0>	Ue	Cif	C5	CSW bus stop number of the core that originated the I/O operation.
rxp0_cif_UeBufNum	Rxp0	Cif	C5	Receive port 0 tells the cache interface which packet buffer number the microengine is working on.
rxp1_cif_UeBufNum	Rxp1	Cif	C5	What buffer is UE working on, in Receive port 1
rxp2_cif_UeBufNum	Rxp2	Cif	C5	What buffer is UE working on, in Receive port 2
copy_cif_UeBufNum	Copy	Cif	C5	What buffer is UE working on, in the Copy port
TaskPending<7:0>	Cif	Ue	C6	Bitmask per microengine thread which tells whether there is 1 or more memory transfer in progress.
TaskFull<7:0>	Cif	Ue	C6	Bitmask per microengine thread which tells whether a thread has already launched the maximum number of memory operations.

Table 5.2: StartIo Interface from Cache Interface to Microengine

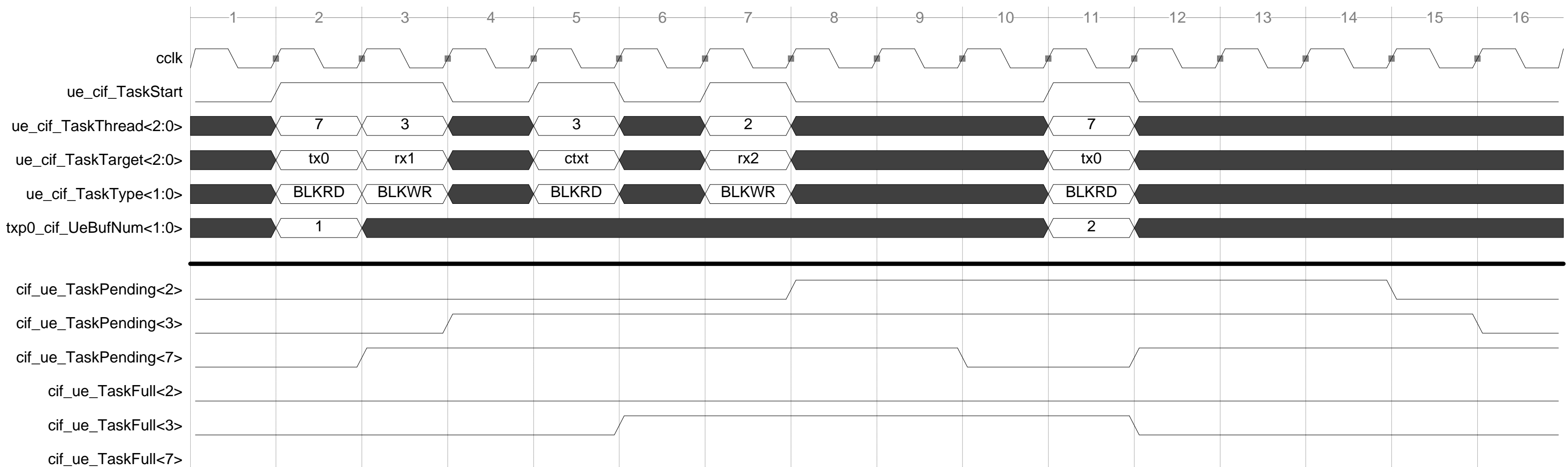


Figure 5.5: Timing of TaskStart Interface

Signal	From	To	Cycle	Description
StartIo	cif	ue	C1	When 1, trigger an I/O read or write microinstruction based on the values on the StartIo signals in the same cycle.
StartIoType	cif	ue	C1	Type of I/O operation. 0=read, 1=write.
StartIoAddr<15:0>	cif	ue	C1	Address of I/O operation. The format is the same as DmaBusAddr, consisting of a unit field and an offset field.
StartIoTid<4:0>	cif	ue	C1	CSW transaction ID of the I/O operation
StartIoOrigin<3:0>	cif	ue	C1	CSW bus stop number of the core that sent this I/O operation
RdyForStartIo	ue	cif	C3	Microengine asserts this whenever it is ready to receive a StartIo operation. The CIF should never raise StartIo unless RdyForStartIo is asserted.

5.2.11.5 Interface to L2 Cache

The cache interface performs four basic types of memory operations: read cache line from memory, write cache line to memory, respond to I/O write from core, and respond to I/O read from core. When reading cache lines, the DMA engine arbitrates for and writes the CmdAddr bus for one cycle to request data from memory. The response may come back many cycles later, so the details of that request are stored in the OutstandingReadTable (ORT). When the response arrives on the incoming Data bus, the OutstandingReadTable tells where the data should be sent within the DMA engine, e.g. transmit port 2 packet buffer at address 0x18. When the data is safely in the packet buffer, the ORT entry is freed so that it can be reused. We support up to 4 outstanding reads at a time. When writing cache lines, the DMA engine arbitrates for and writes the CmdAddr for one cycle, then when a BWTGO comes back, it reads data from the selected internal memory, then arbitrates for and writes the Data bus for four cycles.

Unlike cache line transfers, I/O reads and writes from the cores may arrive at any time in any order. Each core may have a maximum of one I/O request outstanding (as of 4/25/2005), so the cache interface needs a place to store six I/O requests between arrival and completion. For I/O writes, the request arrives on the CmdAddr bus and gets stored in the StartIoQ. The DMA sends a RDIO back to the core, and when the data to be written arrives on the Data bus, it is written to a location in the copy port. The cache interface asks the microengine to execute a special I/O write instruction which reads the data out of the copy port and writes it to the appropriate place inside the DMA engine based on I/O address. I/O reads are implemented in a similar way. The request arrives on the CmdAddr bus and gets stored in the StartIoQ. The cache interface asks the microengine to execute a special I/O instruction which reads the appropriate register or memory and writes the result to the copy port. Then the cache interface arbitrates for and writes the response data from the copy port to the Data bus. The core must not perform more than one outstanding I/O request at a time, or the StartIoQ will overflow; assertions should check that this never happens.

The following sections describe the cycle behavior of the cache interface as it performs several different tasks.

5.2.11.6 Cycle Behavior: TaskStart to CmdAddr Bus

This table describes how a memory transfer request enters the DmaCif through the TaskStart interface and eventually gets driven onto the CSW CmdAddr bus. The cycle numbers start with C5 because that's the stage in the microengine pipeline that the Task is sent to the cache interface.

Stage	Name	Description
C5	Read TaskStart	TaskStart interface decodes the ue_cif_TaskStart signals, decides which queue the task will go into, and prepares to write one of the queues. The inputs to the queues and the write enables are flopped into C6 registers.
C6	Drive CmdAddrReq	Data and control signals for the queues are C6 flops. Data goes into the selected queue module. Each queue also generates an output in C6 (performing bypass if necessary) so that the queue select logic can peek at the head of each queue and decide whether it can issue or not. For example, BRDs cannot issue unless an ORT slot is available. One queue is selected (if any is eligible), and CmdAddrReq is asserted. The TID is provided by the ORT or OWT, which announces the next available slot.
C7	CmdAddrGnt arrives	When CmdAddrGnt arrives, ask the ORT or OWT to fill a slot. The ORT/OWT slot that is filled corresponds to the TID that was driven onto dma_csw_CmdAddrTID in C6. FIXME: Secondary queues FIXME: ort and owt write FIXME: command completion, update counters
C8	ORT/OWT changes	Changes to ORT/OWT appear in C8.
		BELOW IS THE ORIGINAL PIPELINE. PULL ANY USEFUL STUFF OUT, THEN REMOVE IT.
C5	Select Queue	Examine output of each queue and the empty flags. Decide which queue to service next (WriteQ, WriteExtendQ, ReadQ, ReadExtendQ). Only choose from a read queue if there is an empty slot in the OutstandingReadTable. Choose odd/even direction and assert CmdAddrReq.
C1	Arbitrate CmdAddr	Drive the rest of the CmdAddr wires. CmdAddrGnt returns true or false. If true, assert DataReq if needed (reads don't need it) and continue through the pipeline as usual. If false, stall C0 and C1, and continue to assert CmdAddrReq and drive CmdAddr until it is granted once. Meanwhile, begin to read DW01 from DMA internal memory, so that if all goes well we can drive it in C2. NOTE: Arbitration failure in C2 can cause C1 to stall; in this case we must be sure not to bid for CmdAddr after winning it once. DW01 means doublewords zero and one.
C2	Arbitrate Data	At start of C2, the DW01 read is completed, ECC bits is generated, and DW01 data is driven to the cache switch. Later in C2, DataGnt returns true or false. If true, continue through the pipeline as usual and start the read of DW23 so that

5.2.11.7 Memory to DMA Pipeline

Stage	Name	Description
C0	Response Arrives	Response arrives on incoming Data bus
C1	ECC, Dispatch	Check ECC on incoming data and correct single bit errors. Use transaction number as index into OutstandingReadTable, figure out where this data should be written. Prepare to write data to DMA memory.
C2	Write	Write to DMA internal memory at the appropriate location. Clear this slot in the OutstandingReadTable.

5.2.11.8 I/O Access Pipeline (Read and Write)

Stage	Name	Description
C0	Request arrives	I/O read arrives on incoming CmdAddr bus
C1	Store Request	There are 6 IoAccess slots for the 6 cores. Store some of the CmdAddr parameters into the IoAccess slot for the requesting core. If the request is a read, continue through this pipeline. If the request is a write, disable the rest of this pipeline. There is nothing more to do until the Data arrives, one or more cycles later. See I/O Write pipeline below.
C2	Start Uinst	Drive the StartIo interface to the microengine to trigger an IOREAD microinstruction. The I/O read address is sent on cif_ue_StartIoAddr.
C3-C7?	microengine pipeline	IOREAD instruction travels through microengine pipeline. The result is written to registers (per core) in the copy port, then the microengine requests a memory transfer from the copy port address back to the core. The transfer is recorded in WriteQ and processed by the DMA to Memory pipeline, above.

5.2.11.9 I/O Write Pipeline

Stage	Name	Description
C0	I/O write arrival	I/O write data arrives on incoming Data bus.
C1	Read IoAccess	Use the target core number to index into IoAccess and retrieve the CmdAddr portion of the I/O write transaction. Now we have enough information to begin. Prepare to write the data to a register (per core) in the copy port.
C2	Write Copy, Start Uinst	On rising edge of C2, data appears in copy port. Drive the StartIo instruction to trigger an IOWRITE microinstruction. The I/O read address is sent on cif_ue_StartIoAddr.
C3-C7?	microengine pipeline	The IOWRITE instruction travels through microengine pipeline. The instruction reads from the register in the copy port, and the result is written to the address specified by the I/O write request. Then the microengine requests a memory transfer back to the core with a special flag to mark it as an I/O write response. The transfer is enqueued in WriteQ, then enters the DMA to Memory pipeline, above.

5.2.11.10 Task interface pipeline

Stage	Name	Description
C0	TaskStart arrival	TaskStart signal arrives from microengine. Prepare to read MemAddr memory using TaskThread as the address.
C1	Enqueue	Prepare to write memory address and length of transfer into either the Wrq or the Rdq. Compute new values of NumPending registers.
C2	Report	Send Pending/Full status for each thread back to microengine.

5.2.12 Microengine Programming

5.2.12.1 Instructions

The microengine instructions contain the following fields. The microinstruction memory contains DMA_UIM_WORDS words (presently 1024 as of 10/27/05).

Control Field	Bits
Operand A addressing mode	3
Operand A offset	6
Operand B addressing mode	3
Operand B offset	6
Destination addresing mode	3
Destination offset	6
ALU operation	5
Memory transfer	8
Sleep mode	2
Sleep index	4
Branch	4
Next Addr	10
Stall	3
Total	63

The control store needs to be accessible via JTAG; any other path is simply convenience. The DMA engine should be held in reset state (no requests allowed out from cache or switch interfaces) while the control store is being written.

5.2.12.2 Operand selection

Microinstructions need the ability to access certain state variables by special addressing functions. Each of these values must be set up in the thread state before the corresponding variables can be accessed.

The current packet buffer is identified by both the port being serviced and the specific packet to or from that port, which is selected by hardware on a FIFO basis.

5.2.12.3 Destination Selection

TBD

5.2.12.4 ALU operations

In addition to the typical add, subtract, and boolean ALU operations, I imagine some unusual ops, combining two or more "operations" in one opcode because the data required by those operations are all available concurrently. These include calculating address and remaining length in a buffer, queue access, and heap access length checks:

Priority Encode Priority encode looks at operand A bits 31:0 to find the least significant bit that is 1. The result equals the bit number of the least significant bit that is 1. If no bits are set in A<31:0>, the result is zero.

PID Match The ALU compares a 16-bit value taken from bits 31:16 of the packet trailer with a 16-bit field from the Control/Status register file. [?? how to combine comparison test with type dispatch??]

Pointer Update The ALU A operand is a 64-bit value with a physical address in bits 35:0 and a (negative) buffer length in bits 63:36. The B operand is a 28-bit payload length value. The ALU adds the payload length to the address, and adds the payload length to the negative buffer length. The address portion of the A operand (not the sum) is available to the memory address register; the ALU output is available to be written back to the data memory. Branch functions will report whether the buffer length has become positive. This function is used for DMA buffer pointers and queue access.

ALU<35:0> = A<35:0> + B<27:0>
ALU<63:36> = A<63:36> + B<27:0>
Address = A<35:0>

Pointer Distance The operands are pointers, with addresses in bits 35:0. The result has the low 28 bits of their difference in bits 63:36, and the unmodified A operand address in 35:0.

Pointer Extend The result is zero in 63:36, and the difference A<35:0> - B<63:36>, sign extended, in 35:0. This is used for calculating the end of a buffer region.

Offset The A operand is a buffer pointer as in Pointer Update. The B operand is a 28-bit offset in bits 27:0. The ALU adds the B operand to the buffer address, making the sum available to the memory address register. The B operand is compared against the buffer length in bits 63:36 of the A operand. Branch functions will report if the B operand is greater than the buffer length. This function is used for calculating a heap address and checking that the offset is in range.

Swap Offset Like Offset, except that the B operand is in bits 59:32.

Swap Halves A operand bits 31:0 become result 63:32, and B operand bits 63:32 become result 31:0.

Munge The B operand is rotated and masked and or'd according to bits of the A operand. A<5:0> encode a right rotation of the B operand. Bits A<39:8>, are ANDed with bits <31:0> of the rotated value, and bits A<63:40>, are XORed with bits <23:0> of the rotated and anded result. The boolean masks are msb extended with bits 39 and 63, respectively.

Merge0, Merge1, Merge2, Merge3

A merge operation combines operand A and operand B in a programmable way. When loading the microcode, the R_SDmaMergeOpHi/Lo registers are initialized with values that control the behavior of the MergeN instructions. When microcode executes a MergeN instruction, bits from operand A and operand B are combined according to the values in R_SDmaMergeOpHi/Lo. A 1 in the register causes that the corresponding bit will be selected from operand B, while a 0 selects from operand A.

The merge is implemented as follows:

```
For bit from 63 to 32,
  Result[bit] = R_SDmaMergeOpHi[X][bit-32] ? opb[bit] : opa[bit]
For bit from 31 to 0,
  Result[bit] = R_SDmaMergeOpLo[X][bit] ? opb[bit] : opa[bit]
```

5.2.12.5 Sleep Functions

These functions put a thread to sleep (getting no datapath cycles) until a specified event occurs:

Memory transfer completion Wait until a memory transfer has finished and any associated resources can be reused. For writes, this means that the data has been written to the cache switch. For reads, it means that the data is available for the next instruction to use.

Packet buffer available After this instruction, wait until there is a packet buffer available for this thread. This is only defined for txN/rxN/copy port threads which are associated with a port. A receive thread would awaken when a new packet is received from the fabric switch. A transmit thread would awaken when a packet buffer is empty and ready to be build.

Command Arrival After this instruction, wait until a new command arrives from a processor. This would be used in the queue manager, which should not waste cycles polling.

Sleep Forever This instruction causes a thread to sleep indefinitely.

Take Mutex Mutexes are provided so that microengine threads can safely access shared resources such as queues and contexts. A typical scenario is that several threads need to write to an event queue in memory. If all the threads read the queue pointer, write to memory, and write the queue pointer in parallel, then events would get overwritten or lost. Instead, each thread obtains a mutex for the queue, which guarantees exclusive access to the queue pointer and the queue memory. Then the thread reads the queue pointer, writes to memory, updates the pointer, and releases the mutex so that another thread can have its turn.

The Take Mutex function causes the thread to sleep until it owns the mutex identified in the Sleep Index field. The following instruction is allowed to read/write the shared resource, with assurance that no other user of the same mutex is in a critical section. If the mutex is already available, Take Mutex allows the thread to execute again with only the usual delay.

Drop Mutex The "Drop Mutex" function releases a mutex to make it available for other threads. The hardware guarantees that no more than one thread will have ownership of the mutex at any time. The instruction which specifies Drop Mutex is allowed to read/write the shared resource, but any subsequent instructions in the thread must not.

5.2.12.6 Stall

The 3-bit stall field encodes the number of cycles that the dma engine must wait before the current thread is permitted to bid for next use of the datapath. Typically, this field defaults to 1, which ensures that alu results and branch conditions from the current instruction are available for the next. It must be greater than 1 in instructions which issue a memory request or release a packet buffer and wait for it (exact value TBD); it may be zero in instructions whose successor does not depend on any result of the current instruction.

5.2.12.7 Memory Transfer

Microcode needs to be able to initiate and sometimes wait for completion of memory transactions, for packet payloads, queue entries, and buffer and route descriptors. Microcode should be able to specify reads and writes of up to 128 consecutive bytes; reads should be aligned to 8-byte boundaries, writes to 32-byte boundaries. I want to be able to initiate transactions of up to 128 bytes all together, rather than waiting for completion of one 32-byte cache block before starting the next; this may prove complex, and probably results in a different L2 interface for the DMA engine than that used by the processors. Reads and writes of the packet buffers may start at the second or third word of the packet, and are governed by the packet length register.

5.2.12.8 Branch Functions

Some of the branch functions can be arranged to evaluate a small number of bits at the operand register, others must test the alu output. This is important because it determines the branch latency and thus the microinstruction rate of each thread. If we assume that operand access takes one cycle and microinstruction access takes another, that means we can execute instructions from a given thread every second or third cycle.

- Queue pointer test: is queue_pointer + entry_length > queue_limit? (ALU N)
- Buffer descriptor test: is packet_length > buffer_remaining? (ALU N)
- Buffer valid test: is buffer descriptor valid? (combined with above; ALU Z?)
- Type dispatch: PID-Match, Trailer<11:8>
- Queue entry dispatch: decode queue entry
- Port select decode from RDT entry
- Segment length check: ALU<63:32> ≤0 (ALU N)
- Context match: Aop<31:16> = Bop<31:16> (ALU Z)
- Sequence number match: Aop<63:32> = Bop<63:32> (ALU Z)
- Queue empty? full (room for more)? (ALU N)

- Completion notification?
- Error detected?

I do not presently see a need for subroutines, especially if we have a general case dispatch for port select and queue entry decode.

There's a question how a thread sleeps when it's waiting for memory or an available packet buffer or a mutex with another thread.

5.2.12.9 Next Address

In the absence of a branch function, each microinstruction address is the contents of the NextAddr field of the previous microinstruction of the same thread. Branch functions substitute conditional values for some of the low-order NextAddr bits.

5.2.13 Unified Engine

There are several splits one could imagine in the DMA Engine; I have chosen a unified approach because I wanted to build sufficient capacity to saturate the pin bandwidth at the memory and switch interfaces, and I wanted that capacity to be available to any thread that needed it. I am hopeful that the transmit side and the receive side can each be implemented as a pipelined microengine with four threads serviced on a demand round-robin basis. Branching instructions (virtually all) would probably require a latency of two cycles: (cycle 1) Branch address computation, microinstruction fetch; (cycle 2) read operands; (cycle 3) run alu, write result. Bypass is not necessary, except perhaps for global variables, because branch prohibits executing the same thread next cycle.

5.2.14 Bandwidth

The available main memory bandwidth, assuming two ports of 400 MHz DDR2 memories 8 bytes wide, is 6.4 GB/sec, rising to 12.8 as DDR data rates rise to 800 MHz. The L2 cache can deliver 16 GB/sec on hits, but every miss costs two L2 cycles. As a result, with full memory bandwidth demand there is only 3.2 GB/sec available for hits, so the available bandwidth drops to 9.6 GB/sec when the memory is busy. [I'd really like to improve the L2 bandwidth, if we can do that cheaply. Easiest change seems to be to bury writes under reads of the opposite half-line.]

The possible demand from the switch consists of 6 ports at 2 GB/sec, derated by the 8/10 code, resulting in 10 GB/sec; but realistic load conditions further derate that demand by factors of 1/5.5 (to account for average path length) and 88% (payload fraction of packet size), to about 1.6 GB/sec with uniformly distributed traffic. For communication between two nodes on an otherwise idle fabric, we should be able to sustain 4.2 GB/sec of payload delivery.

In the point-to-point DMA case, we will have three input or output ports running at full bandwidth, each transfering a packet every 95 ns, more or less. This means that in a fully-unified DMA engine no stage in the pipeline should dedicate more than 32 ns, or 8 cycles, to one packet; that means the payload cannot be copied – it must be transferred directly between the switch buffer and the cache.

5.2.15 Matching

Content-addressable memory is expensive in power and area, so we'd prefer not to keep large numbers of receive contexts ready to match each incoming packet. Still, we do need to be able to keep several contexts open at a time; the compromise is to build the equivalent of a direct-mapped cache. A few bits of the context-id are used to index the receive context space, and the remaining bits compared with the id stored at that location. If there is no match, the packet is assumed to be a remnant of an aborted transfer and will be discarded; we'll count such events. Similarly, each received packet carries a 16-bit process id and a 4-bit process index in the trailer; the process index is used to address the memory which holds control/status pages, and the selected page is checked to match the process id, which must match to use the process variables. Process index 15 is reserved for non process-specific packets, and indexes 12-14 are reserved for global variables.

5.2.16 Interface registers

The DMA Engine needs controls for both user and kernel threads, and we need to protect some of the controls of the user space from access by user mode. Still, we can use a pair of pages for each processor, one user-writable and the other only writable by the kernel to define the required variables for both threads.

It's also important to think carefully about which information needs to be in interface registers, which are presumably uncachable, and which should be in cacheable memory, where we can use more efficient cache-block transfers. [We could implement the control/status registers in such a way that they were cacheable; we'd just have to remember ownership, and invalidate the owner whenever one changed.]

5.2.17 Coherence

We clearly need a coherent model of the interface between software and the DMA Engine control interface. We also need to ensure that accesses by concurrent segments see a coherent memory interface, whether tested by streams from different remote nodes or by overlapping stride patterns.

I am hoping that we can achieve coherence with the processors by means of a simple interlock which a thread grabs when it reads memory for modification, and releases upon write. The interlock should delay intervention responses, and if we make its use exclusive among threads, it will ensure coherence among threads. I still need to explain how a thread waits for the lock to become available; can we inhibit its bid for cycles?

5.2.18 Alignment

For efficient operation of the DMA engine, both transmit and receive buffers should be aligned on 8-byte boundaries and transferred data sizes should be multiples of 8. There are at least three distinct performance levels: highest performance is achieved using contiguous data in large buffers aligned to 64-byte boundaries; the DMA engine can achieve intermediate performance with efficient transfers of data in multiples of 8 bytes, aligned to 8-byte boundaries. Transfers that do not meet these criteria must be handled by software, and suffer significantly higher penalties.

5.2.19 Strides and Scatter/Gather

For aligned contiguous transfers, the DMA engine has a substantial performance advantage over the processor cores, in that it can copy entire cache blocks at a rate of one per cycle (8 GB/sec). This advantage disappears in strided or scatter/gather operations, where each packet may require multiple memory references and must be assembled and disassembled piecemeal. With a reasonable datapath width (say, 8 bytes), the peak transfer rate falls to 2 GB/sec, the same as the peak copy rate of a 5Kf core. The DMA engine still has the potential of substantially reduced overhead, because of having been designed specifically for the purpose, but that shows up as overhead hardware (parallel 32-bit adders, for example). A software implementation, conversely, would have the option of defining special cases to eliminate some of the overhead.

If we get rid of sub-block access in the DMA engine, we'll also force Enq_* and Wr_Heap to cache block sizes.

5.2.20 Output Thread

There is an output thread associated with each output port of the switch, and one with the copy function. Each such thread has at least three output buffers, and the thread works on setting up one, while the cache is filling the second, while the third is being emptied by the switch. When it finishes setting up the cache requests to fill a buffer, the thread waits for availability of the next buffer. If there is a ready transmit context for this output port, it builds a packet in the buffer, enables it for output, and returns to the top. See the pseudocode (**??**) for a more detailed flow.

The first microinstruction of an output thread is executed with process variables and transmit context selected. It copies the routing information from the transmit context to the packet header. The second writes the Packet Type and Process ID to the trailer ford. It tests whether the context is for a DMA packet, and if not, whether the payload comes from the queue entry or the heap.

If the payload is from the queue entry, the third microinstruction loops copying payload to the packet buffer (or loads it from queue memory).

If the payload is from the heap, the third microinstruction checks the heap offset and length, and the fourth reads from the heap to the packet buffer.

If the packet is DMA, the third microinstruction checks the BDT entry; the fourth initiates a memory read, updates address and length, and checks the BDT again; the fifth initiates a second read and checks the segment length. If the segment is done, the sixth microinstruction pops a new transmit context from the appropriate queue.

5.2.21 Input Thread

There is an input thread associated with each input port of the switch. Each such thread rotates among three input buffers, making one available to the switch as it interprets the control information in the next, and the cache stores the payload of the third. Upon completing a packet buffer, it waits for the next to be full, then finds the receive context that matches this packet. Finding one, it sets up the cache requests to store the payload according to the context, writes any event queue entry required, writes any response queue entry required, and returns to the top. If there is no matching receive context, the input packet is discarded. See the pseudocode (**??**) for a more detailed flow.

For each received packet, the first microinstruction executed is selected by the packet type field, and the hardware uses the 4-bit process index to select one of the control/status pages to control heap and queue accesses. In all cases, the first instruction checks that the PID matches that in the selected C/S page.

The first cycle of an Enq_Tx, _Rx, or _Direct tests the queue pointers for the appropriate queue to make sure there is room for the new packet. The next cycle either writes the packet to the queue (one or two writes required) while updating the pointers; wraps the queue pointer if needed, then writes the packet; or sets an error indication because the queue has overflowed.

The first cycle of a Wr_Heap tests the offset and length against the heap size. The second cycle either writes the payload or sets an error indication because the store is out of bounds.

The first cycle of a DMA selects the receive context and checks for a match; it checks for a packet sequence number match, and it checks that the current BDT entry has room for the first cache line of the packet. The second cycle writes the first cache block, increments the address, decrements the buffer length, and checks that the BDT entry has room for the second cache block. The third cycle writes the second block, increments the address, decrements the length, and checks whether the message segment is complete, and whether a notification is needed. If it is, the fourth cycle pushes the Ack onto the transmit foreground queue. In either case, upon completion of a segment, a new receive context is popped off the receive queue.

5.2.22 Thread performance

There are tight performance constraints on packet processing: in point-to-point communication, a full-sized DMA packet may arrive every 95 ns on each input port (152 bytes * 10 bits/byte / 16 bits/ns). Assuming a 250 MHz (4ns) clock in the DMA engine, we have 24 cycles in which to service 3 packets, or 8 cycles per packet. To achieve this goal, we'll need to be very efficient in our use of cycles, especially in dispatching to the appropriate routine for each case.

The first microinstruction can be selected by the hardware on the basis of packet type and validity. Subsequent microinstructions can be pipelined to select register file operands, alu operations, and branch decision before fetching the next microinstruction of the same thread.

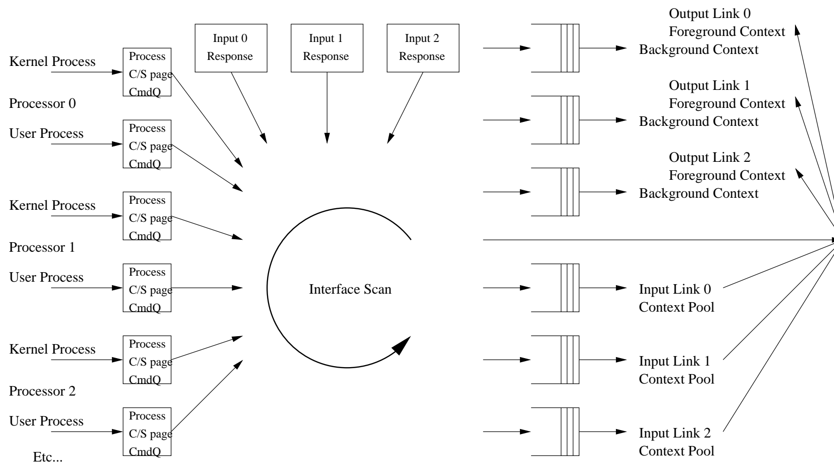
Short-message latency imposes an additional constraint: of 500ns total, the fabric requires about 180, leaving 320ns for DMA Engine and library software. We want to make sure that when the Tx command queue is empty, new entries are passed directly to the output thread without diverting through memory, and received packets are available on the event queue with absolutely minimal overhead.

5.2.23 Queue manager

The queue manager state machine is activated whenever a new entry is stored in the command registers or an input thread has something to enqueue for response. It checks the queue entry for validity (allowed buffer and route descriptors) and copies it to appropriate memory for access by the input and output threads. A related state machine writes event queue entries. Figure 5.6 is a schematic representation of the queue Manager process for transmit and receive queues.

When a transmit context is completed, the queue manager pops the next item off the transmit queue associated with the same output port. When a receive context is completed, the queue manager pops the next item off the receive queue associated with the input port, and assigns a new context id. It inserts the new context id into a Get request (the rest of which was in the receive queue entry), and passes the Get request to the bypass transmit context of the output port selected by the route.

Figure 5.6: Queue manager



5.2.24 Port manager

Each input and output port has a state machine which manages three or more packet buffers, such that one of them is assigned to the switch port and is used for injecting a packet into the fabric or receiving one from the fabric; one of the packet buffers is assigned to the cache, and sequences the transfer of up to two aligned L2 cache lines to or from the cache, using an address provided by the DMA engine; the third packet buffer is assigned to the port's input or output thread in the DMA engine, which can read or write it under microcode control. The roles assigned to the three buffers rotate when all have completed their respective tasks, or have nothing to do. [It might pay to have four packet buffers per port, so that variations in processing time can be absorbed without degrading performance.]

5.2.25 Copy Thread

There is also a low-priority thread which performs memory-to-memory transfers; it appears that the simplest implementation treats the copy thread as an additional input and output port which software can treat as if it were simply another interface to the fabric with a loopback destination. I'd like to augment the copy function with a useful crypto function (for encryption/decryption of TCP/IP traffic) and a zero-memory function (for use by the page-creation software). And if the fabric processor is going to be responsible for strided and scatter/gather operations, it would be helpful if the copy thread could be invoked to prefetch memory along strided or scattered streams.

5.2.26 Timeouts

We need to be able to detect lost packets or broken links without incurring significant software overhead. One way to do that would be to set a timer on each active receive context, and complain about any that exceeded a software-defined maximum value, either since initialization, or since the last received packet. Transmit contexts may not benefit, though it would be desirable to invoke software if unable to emit a packet over some defined time period.

Timers on receive contexts won't detect lost Acks or Rendezvous requests, nor anything which doesn't have a dedicated receive context.

5.2.27 Error Conditions

- Correctable ECC error on memory access: correct the error, capture address and syndrome in error registers.
- Uncorrectable error on memory access: disable the thread with the error, capture address, interrupt host.
- Receive packet with "poison" type: count and drop packet
- Receive packet with process id mismatch: enqueue for fabric processor?

- Receive DMA packet with context id mismatch: count and drop packet
- Event queue overflow: Interrupt
- Any command queue overflow: Interrupt
- Route or Buffer handle out of bounds: Interrupt
- Invalid Buffer Descriptor: Interrupt, push current context onto event queue, invalidate context.

5.3 Notes

5.3.1 Rendezvous

Rendezvous performance is pretty important, because any latency becomes message overhead, so we'd like it to be handled in hardware if at all possible. The rendezvous response, when it returns, should start the dma transfer, including setting up any shifting necessary to get the packets aligned to cache blocks at the destination, and setting the segment packet count to stop at a page boundary.

The communicator data structure can include an array indexed by rank, where each item in the array is the head of an ordered list of posted receives, and there can be a separate list of receives for rank-any. The structure representing a posted receive could include an sequence number, so we could determine which is oldest. But are we then going to hash on the tag?

Rendezvous is also responsible for calculating the alignment of segments to ensure that packet payloads are aligned to cache blocks at the receiver. [The transmitter is in a much better position than the receiver to do the alignment, because the packets may arrive at the receiver from multiple interleaved streams. The receiver would also have to do read-modify-writes if it got partial lines.]

Therefore, the rendezvous request includes the low bits of the source buffer starting address and the buffer length. The rendezvous response carries (in the datatype field) the number of bytes by which to shift source cache blocks to align them with destination cache blocks, and the length of the segment. I intend the rendezvous response to be coded as a Get: the relevant parameters are in the transmit context to initiate the transfer.

DMA transfers always reserve a receive context before queueing for use of a transmit context; this prevents a potential deadlock which could occur if some transfers reserved transmit contexts first, then queued waiting for receive contexts.

5.3.2 Ethernet simulation

We'd like to be able to pretend that the fabric is one large ethernet, so as to use TCP and UDP services with a minimum of new development. For that purpose, there should be a driver with the same API as the ethernet driver, but which converts MAC addresses to routes through the fabric or broadcast.

5.3.3 Barrier

When any communicator is created, the data structure in each node includes space reserved for barriers and collective operations on that communicator. The nodes which participate in the communicator are partitioned into a tree or multidimensional network which will be used for barriers and collectives. The data structure in each node describes where this node is in that network. Specifically, how many inputs are required at this point in the network to complete a barrier, and where to send notification of barrier stage completion.

If we have explosive broadcast, I expect that a collection tree followed by broadcast notification will be the most efficient implementation of barriers. Without it, we may find that a single-pass multidimensional exchange works better, in spite of needing more messages.

In the two-pass tree-structured implementation, most nodes are leaves, some are intermediate, and one (arbitrarily chosen, from the perspective of the MPI user) is the root. Leaf nodes, upon encountering a barrier, send a packet containing the communicator id to their designated intermediaries, which mark receipt from each leaf and the local process in the communicator data structure. Upon receipt of the last notice, an intermediate node sends notice to its designated superior, in just the way that each leaf did to the intermediate node. The root, rather than sending notice to a superior, responds with a broadcast which notifies all ranks of the communicator that the barrier has been passed. Intermediate nodes must then reset the communicator data structure in preparation for the next barrier. [This seems to imply a race. Maybe barriers and collectives should carry a generation number.]

If we use a single-pass implementation, there is no distinction between leaf, intermediate, and root nodes. The communicator is factored by some small integer radix r, and each node exchanges messages with $r-1$ other nodes at each stage of the barrier process. The barrier is complete after k stages, where $r^k \geq N$, the size of the communicator.

5.3.4 Cache interface

If we provided a single 8-byte wide interface to the L2 cache, operating at 250 MHz, the peak achievable bandwidth would be 2 GB/sec, only half of the goal. I think the answer is that the buffers for each input or output port should interface a 64-bit bus with its own path to or from the L2. Each buffer needs to be able to handle out-of-order completion of reads, because some will be found in cache while others are in memory.

Writing toward memory is easier because the data transfer will occur at a fixed time with respect to successful arbitration and address transfer, so the packet buffer can schedule each write as the last one is completing, and the path from a packet buffer to the cache data bus simply carries one doubleword per cycle.

We will require receive packet payloads to be aligned with the memory at 32-byte boundaries, corresponding to a half line in the L2 cache. This eliminates the need to read a block from memory before writing the payload over it.

5.3.5 Performance Counters

It's important to be able to measure and understand the performance of the communication fabric and to be able to characterize the load presented by an application. Some such data can be gathered by library software, noting initiation and completion times of messages, their lengths and other characteristics. But some information is undoubtedly best obtained by instrumenting the hardware. So far, I don't know what to measure in the DMA engine.

- For measurement of traffic through the switch, I'd like to have a performance sampling bit in the packet header, and accumulate a total of the time spent in the switch for all packets with the sampling bit set. An implementation of that function would attach a packet arrival time to every packet, and calculate the running sum of the difference between arrival and departure times for packets with the bit set.
- I'd also be interested in knowing the number of cycles in which each virtual channel is blocked (has no apparently free buffers in the downstream node).
- Packets received/transmitted per DMA engine port (by type?)
- Memory ECC errors (count, or simply flag?)

5.4 Registers and Definitions

[$Id: DmaRegs.lyx 46805 2007-10-30 21:33:40Z denney $]

5.5 Microengine Instructions

5.5.1 Instruction Fields

Class

DmaUeInst

Attributes

Bit	Mnemonic	Type	Reset	Definition
d0[2:0]	opaMode	DmaUeInstOpa		operand A addressing mode
d0[8:3]	opaIdx			operand A index
d0[11:9]	opbMode	DmaUeInstOpb		operand B addressing mode
d0[17:12]	opbIdx			operand B index
d0[20:18]	destMode	DmaUeInstDest		addressing mode for destination
d0[26:21]	destIdx			destination index
d0[31:27]	alu	DmaUeInstAlu		ALU operation
d0[34:32]	memOp	DmaUeInstMemOp		memory transfer operation
d0[35]	memWrAddr			write MemAddr register for current thread
d0[38:36]	memLenSel	DmaUeInstMemLenSel		specifies where the memory transfer length comes from, either a constant or from the payload length in a port.
d0[39]	memLast			For threads 0-7, memLast=1 means "This is the last instruction that refers to this packet." If 1, the currently selected port is notified that the microengine is finished with the packet buffer. In the DMA_THR_IO_ACCESS thread, memLast=1 informs the StartIo interface that the microengine is ready for another I/O operation. For I/O reads this causes the cache interface to send the I/O data back to the processor. In the I/O thread, when memLast=1, the memOp must encode NONE, sleepMode must encode hwFlag, and sleepIndex must encode NONE.
d0[41:40]	sleepMode	DmaUeInstSleep		sleep request
d0[45:42]	sleepIndex			which condition or mutex is indicated in sleep field. The encoding is DmaUeInstSleepCond or DmaUeInstSleepMutex, depending on sleepMode
d0[49:46]	branch	DmaUeInstBranch		branch type
d0[59:50]	nextAddr			next address
d0[62:60]	stall		1	number of cycles to delay before this thread may execute another instruction
d0[63:0]	allBits			for vspecs to read the first doubleword as a bit vector. Overlaps allowed.

5.5.2 Operand A addressing modes

This section describes the values that can go into the opaMode field of the microengine instruction.

Enum

DmaUeInstOpa

Attributes

-allowlc

Constant	Mnemonic	Definition
3'd0	specialReg	Special Operand A registers, see table below
3'd1	ptr0	Read from dmem. The 10-bit address is ptr0<9:0> xor (opaIdx<5:0> shifted left by 4).
3'd2	ptr1	Read from dmem. Same as above, but with ptr1.
3'd3	ptr2	Read from dmem. Same as above, but with ptr2.
3'd4	ptr3	Read from dmem. The 10-bit address is ptr3<9:0> xor (opaIdx<5:0> shifted left by 4) xor processIndex<3:0>. For threads 0-3, the processIndex comes from bits 15:12 of the trailer FORD in the selected receive port. For the I/O thread, the processIndex<3:0> comes from the I/O address bits 19:16.
3'd5	ptr4	Read from dmem. The 10-bit address is ptr4<9:0> xor (opaIdx<5:0> shifted left by 4).
3'd6	ptr5	Read from dmem. The 10-bit address is opaIdx<5:0> concatenated with 1111.
3'd7		reserved

5.5.3 Operand B addressing modes

This section describes the values that can go into the opbMode field of the microengine instruction.

Restriction: The dmem is divided into four banks, and each bank can only read one address at a time. If operand A and operand B select different addresses in the same dmem bank, operation is undefined.

Enum

DmaUeInstOpb

Attributes

-allowlc

Constant	Mnemonic	Definition
3'd0	specialReg	Special Operand B registers, see table below
3'd1	ptr0	Same as ptr0 in operand A except opbIdx is used.
3'd2	ptr1	Same as ptr1 in operand A except opbIdx is used.
3'd3	ptr2	Same as ptr2 in operand A except opbIdx is used.
3'd4	ptr3	Same as ptr3 in operand A except opbIdx is used.
3'd5	ptr4	Same as ptr4 in operand A except opbIdx is used.
3'd6	ptr5	Same as ptr5 in operand A except opbIdx is used.
3'd7	memRead	Thread-specific buffer for microcode to read cache blocks. The buffer can be filled using the memory fields of the instruction. There are 16 doublewords of data in the buffer, selected by opaIdx<3:0>.

5.5.4 Destination Addressing Modes

Enum

DmaUeInstDest

Attributes

-allowlc

Constant	Mnemonic	Definition
3'd0	specialReg	Special destination registers, see table below
3'd1	ptr0	Same as ptr0 in operand A except destIdx is used
3'd2	ptr1	Same as ptr1 in operand A except destIdx is used
3'd3	ptr2	Same as ptr2 in operand A except destIdx is used
3'd4	ptr3	Same as ptr3 in operand A except destIdx is used
3'd5	ptr4	Same as ptr4 in operand A except destIdx is used
3'd6	ptr5	Same as ptr5 in operand A except destIdx is used
3'd7	memWrite	Thread-specific buffer for microcode to write cache blocks, indexed by destIdx. The buffer can be sent to memory using the memory fields of the instruction. There are 16 doublewords of data in the buffer, selected by destIdx<3:0>.

5.5.5 Special Registers addressed by Operand A

Enum

DmaUeInstSpecialOpa

Attributes

-allowlc

Constant	Mnemonic	Definition
6'h00	zero	The value zero.
6'h10	thread0Ptr	Read THREAD0_PTR register, pointer state for the Rx port 0 thread. This is used to implement I/O reads of thread state registers. Also, it allows a thread to read its ptrN values.
6'h11	thread1Ptr	Read pointer state for Rx port 1
6'h12	thread2Ptr	Read pointer state for Rx port 2
6'h13	thread3Ptr	Read pointer state for Rx copy port
6'h14	thread4Ptr	Read pointer state for Tx port 0
6'h15	thread5Ptr	Read pointer state for Tx port 1
6'h16	thread6Ptr	Read pointer state for Tx port 2
6'h17	thread7Ptr	Read pointer state for Tx copy port
6'h18	thread8Ptr	Read pointer state for Queue Manager
6'h19	thread9Ptr	Read pointer state for I/O service thread: THREAD9_PTR register
6'h1E	spclData	Returns the 24-bit data from the most recent SPCL operation that arrived on the CSW. This register is only used in microcode that handles SPCLs. To compute spclData, concatenate 40 zeroes, ioAddr<35:20>, and ioAddr<15:8> to make a 64-bit value.
6'h1F	ioAddr	Returns the address of the current I/O read or write. This is used in microcode that implements programmable I/O reads or writes.

5.5.6 Special Registers addressed by Operand B

In this table, the constant represents the value to be used in opbIdx when the microinstruction reads special registers. To read a special register, set opbMode to SPECIAL_REG. The registers whose names start with "io" are used to implement I/O reads and should not be used in normal microcode.

Enum

DmaUeInstSpecialOpb

Attributes

-allowlc

Constant	Mnemonic	Definition
6'h00	zero	The value 0.
6'h1F	ioData	Returns the data of the current I/O write. This is used in microcode that implements programmable I/O writes.
6'h20	pktHead	Read the packet header FORD for the currently selected receive port. The receive port number is based on the thread number.
6'h24	pktCtl	Read the packet control FORD for the selected receive port. If there is no control FORD, according to the hasCtrl bit in pktHead, the pktCtl register retains its value from the last packet that did have a control FORD.
6'h28	pktTrail	Read the packet trailer FORD
6'h2C	pktLen	Read the packet payload length for the selected receive port, in units of bytes. The payload length may be between 8 and 128 bytes, but always a multiple of 8. The header, control, or trailer FORDs are not counted as payload.
6'h30	pktPayload0	Read the first doubleword of payload in the currently selected receive port. Continues until...
6'h31	pktPayload1	Second dw of payload
6'h32	pktPayload2	Third dw of payload
6'h33	pktPayload3	Fourth dw of payload
6'h34	pktPayload4	Fifth dw of payload
6'h35	pktPayload5	Sixth dw of payload
6'h36	pktPayload6	Seventh dw of payload
6'h37	pktPayload7	Eighth dw of payload
6'h38	pktPayload8	Ninth dw of payload
6'h39	pktPayload9	Tenth dw of payload
6'h3A	pktPayload10	Eleventh dw of payload
6'h3B	pktPayload11	Twelfth dw of payload
6'h3C	pktPayload12	Thirteenth dw of payload
6'h3D	pktPayload13	Fourteenth dw of payload
6'h3E	pktPayload14	Fifteenth dw of payload
6'h3F	pktPayload15	Read the sixteenth doubleword of payload in the currently selected receive port.

5.5.7 Special Registers addressed by Destination

In this table, the constant represents the value to be used in destIdx when the microinstruction writes special registers. To write a special register, set destMode to SPECIAL_REG. The registers whose names start with "io" are used to implement I/O reads and should not be used in normal microcode.

Enum

DmaUeInstSpecialDest

Attributes

-allowlc

Constant	Mnemonic	Definition
6'h00	trashcan	Null destination; used when an instruction does not write any result.
6'h08	ptr0	Write ptr0 register in thread state from bits 9:0 of the ALU result. If the modified pointer is used in the next instruction, the instruction that writes the pointer must set Stall to at least 3 to avoid a pipeline hazard.
6'h09	ptr1	Write ptr1 register. See pipeline hazard warning above.
6'h0A	ptr2	Write ptr2 register. See pipeline hazard warning above.
6'h0B	ptr3	Write ptr3 register. See pipeline hazard warning above.
6'h0C	ptr4	Write ptr4 register. See pipeline hazard warning above.
6'h0F	ioData	Write ALU result to the I/O read response buffer. This is used in microcode that implements a programmable I/O read.
6'h20	pktHead	Write packet header FORD for the currently selected transmit port. The transmit port number comes from the portSel field in thread state. The NumFords field of the header is calculated as the payload length in fords plus 2 (header and trailer) plus 1 if HasCtrl is set. The payload length must be set before writing the header.
6'h24	pktCtl	Write packet control FORD for the selected transmit port. The control FORD is only written to the fabric switch if the hasCtrl bit in pktHead is set.
6'h28	pktTrail128	Write packet trailer FORD, setting the packet payload length to 128 bytes.
6'h2C	pktTrailLen	Write packet trailer and payload length for the currently selected transmit port. The payload length is taken from bits 7:0 of the alu, must be between 8 and 128, and always a multiple of 8. If hasCtrl=0 in the header, payload length must be between 16 and 128 bytes. The header, control, or trailer FORDs are not counted as payload. The length must be set before writing the packet header.
6'h30	pktPayload0	Write the first doubleword of payload in the currently selected transmit port. NOTE: When writing any of the pktPayloadN registers, you must **ensure that any outstanding memory reads from the previous packet (before memLast) have completed**; otherwise the current AND previous packet may be corrupted. See bug 2297 for further analysis.
6'h31	pktPayload1	Second dw of payload
6'h32	pktPayload2	Third dw of payload
6'h33	pktPayload3	Fourth dw of payload
6'h34	pktPayload4	Fifth dw of payload
6'h35	pktPayload5	Sixth dw of payload
6'h36	pktPayload6	Seventh dw of payload
6'h37	pktPayload7	Eighth dw of payload
6'h38	pktPayload8	Ninth dw of payload
6'h39	pktPayload9	Tenth dw of payload
6'h3A	pktPayload10	Eleventh dw of payload
6'h3B	pktPayload11	Twelfth dw of payload
6'h3C	pktPayload12	Thirteenth dw of payload
6'h3D	pktPayload13	Fourteenth dw of payload
6'h3E	pktPayload14	Fifteenth dw of payload
6'h3F	pktPayload15	Write the sixteenth doubleword of payload in the curretnly selected transmit port.

5.5.8 ALU Operation Field

Enum

DmaUeInstAlu

Attributes

-allowlc

Constant	Mnemonic	Definition	(N)	(Z)	Produc
5'd0	selA	select A operand (A+0)	A<63>	A<63:0>	
5'd1	selB	select B operand (0+B)	B<63>	B<63:0>	
5'd2	add	A + B	sum<63>	sum<63:0>	
5'd3	sub	A - B	dif<63>	dif<63:0>	
5'd4	boolAnd	boolean AND (A & B)	AND<63>	AND<63:0>	
5'd5	boolOr	boolean OR (A | B)	OR<63>	OR<63:0>	
5'd6	boolXor	boolean XOR (A ˆ B)	XOR<63>	XOR<63:0>	
5'd7	boolAndN	boolean ANDNot (A & ˜B)	ANDN<63>	ANDN<63:0>	
5'd9	priorityEncode	result<4:0> = priority encode of A<31:0>. The result is the bit number of the lowest bit of A that is set, or zero . upper result bits are 0.	result<4>	A<31:0>	
5'd10	pidMatch	compare 16-bit value from bits 31:16	0	XOR<31:16>	
5'd12	ptrUpdate	Pointer Update: memAddr = A<35:0>; Alu<35:0> = A<35:0> + Zext B<27:0>; Alu<63:36> = A<63:36> + B<27:0>	alu<63>	alu<63:36>	
5'd13	ptrDist	Pointer Distance: Alu<35:0> = A<35:0>; Alu<63:36> = A<27:0> - B<27:0>	alu<63>	alu<63:36>	
5'd14	ptrExtend	Pointer Extend: Alu<35:0> = A<35:0> - Sext B<63:36>	alu<63>	alu<63:36>	
5'd15	offset	calculate heap address and check offset. Alu<63:36> = A<63:36> + B<27:0>; Alu<35:0> = A<35:0> + Zext B<27:0>	alu<63>	alu<63:36>	
5'd16	swapOffset	calculate heap address and check offset. Alu<63:36> = A<63:36> + B<59:32>; Alu<35:0> = A<35:0> + Zext B<59:32>	alu<63>	alu<63:36>	
5'd18	subLow32	Subtract in low 32 bits only. Result<31:0> = A<31:0> - B<31:0> Result<63:32>=A<63:32> N = Result<31> Z based in Result<31:0> only			
5'd20	cacheRead	result = B<63:0>	0	MemAddr<35:0> xor A<35:0>	
5'd21	cacheWrite	Write two dmem locations. B<63:0> is the alu result, and is written to destination address, which must have bit 4 = 1. MemAddr<35:0> is written to destination address minus one.	alu<63>	result<63:0>	
5'd23	munge	Rotate/and/xor operations on Operand B, controlled by bits of Operand A. First rotate right by opa<5:0>. Any bits that shift off the end wrap around. Then AND with opa<39:8> (msb extended with opa<39>). Then XOR with opa<63:40> (msb extended with opa<63>).	B<63>	result<63:0>	
5'd24	merge0	Twice9 only: Merge A and B, based on bits from R_SDmaMergeOpHi[0] and R_SDmaMergeOpLo[0]. See 5.2.12.4 for details.	alu<63>	result<63:0>	TWC9
5'd25	merge1	Twice9 only: Merge A and B, based on bits from R_SDmaMergeOpHi[1] and R_SDmaMergeOpLo[1].	alu<63>	result<63:0>	TWC9
5'd26	merge2	Twice9 only: Merge A and B, based on bits from R_SDmaMergeOpHi[2] and R_SDmaMergeOpLo[2].	alu<63>	result<63:0>	TWC9

5.5.9 Memory Operation Field

Enum

DmaUeInstMemOp

Attributes

-allowlc

Constant	Mnemonic	Definition
3'b000	none	Don't start any memory operation
3'b001	memRead0	Start block read from memory to the thread's Memory Read Buffer, doublewords 0-7. The memory address comes from the MemAddr register for the thread.
3'b010	memRead1	Start block read from memory to the thread's Memory Read Buffer, doublewords 8-15. The memory address comes from the MemAddr register for the thread.
3'b011	readPkt	Start block read of currently selected packet buffer
3'b100	sendIntr	Send an interrupt to a processor. The instruction that sets sendIntr must produce an alu result in which result<15:12> is the bus stop number of the interrupt target and result<11:0> is the unique number that goes on CmdAddr.
3'b101	memWrite0	Start block write from the thread's Memory Write Buffer, doublewords 0-7, to memory. The memory address comes from the MemAddr register for the thread.
3'b110	memWrite1	Start block write from the thread's Memory Write Buffer, doublewords 8-15, to memory. The memory address comes from the MemAddr register for the thread.
3'b111	writePkt	Start block write from the currently selected packet buffer.

5.5.10 Memory Transfer Length Selection

Enum

DmaUeInstMemLenSel

Attributes

-allowlc

Constant	Mnemonic	Definition
3'b000	payloadLen	Transfer length comes from the payload length of the port associated with the current thread. Only threads 0-7 may use this encoding.
3'b001	bytes8	Use transfer length of 8 bytes. The hardware will transfer a whole cache block, but the thread may be able to awaken sooner than if it asked for a 64 byte transfer.
3'b100	bytes32	Use transfer length of 32 bytes
3'b010	bytes64	Use transfer length of 64 bytes
3'b101	bytes96	Use transfer length of 96 bytes
3'b011	bytes128	Use transfer length of 128 bytes

5.5.11 Sleep Mode Field

Enum

DmaUeInstSleep

Attributes

-allowlc

Constant	Mnemonic	Definition
2'b00	hwFlag	After this instruction, sleep until a certain hardware flag is detected, for example the completion of a memory transfer. The condition is determined by the Sleep Index field.
2'b01		Reserved
2'b10	takeMutex	After this instruction, sleep until this thread has exclusive ownership of the mutex identified in the Sleep Index field. The following instruction is allowed to read/write the shared resource.
2'b11	dropMutex	After this instruction is completed, release the shared resource identified in the Sleep Index field. The instruction which specifies DropMutex is allowed to read/write the resource, but the following instruction must not.

5.5.12 Sleep Index Field, when Sleep=HwFlag

If the Sleep field equals HwFlag, the Sleep Index field is encoded as follows:

Enum

DmaUeInstSleepFlag

Attributes

-allowlc

Constant	Mnemonic	Definition
4'b0000	none	Don't sleep. For most instructions, you don't want a sleep operation, so you should encode NONE.
4'b0001	halt	Halt is a sleep flag that is always false. If a process sleeps on this flag, it will never wake up. The only way a thread can awaken from halt is external software modification of the thread state.
4'b0010	buffer	After this instruction, sleep until there is a full (Rx) or empty (Tx) packet buffer from the selected port. The port that is monitored for packets is determined by the thread number. If SleepIndex=buffer is used in the same instruction as memLast=1, the stall field must contain at least 5 (?).
4'b0011	mem	After this instruction, sleep until all memory transfers started by previous microinstructions on this thread have completed. If SleepIndex=mem is used in the same instruction as starting a memory operation, the Stall field must contain at least 4.
4'h4-4'hF		Reserved

5.5.13 Sleep Index Field, when Sleep=TakeMutex or DropMutex

If the Sleep field is TakeMutex or DropMutex, the Sleep Index field tells which Mutex the instruction tries to acquire. The field is encoded as follows:

Enum

DmaUeInstSleepMutex

Attributes

-allowlc

Constant	Mnemonic	Definition
4'd0	ptr0	Use ptr0 bit 8 concatenated with bits 3:0 to select the mutex.
4'd1	ptr1	Use ptr1 bit 8 concatenated with bits 3:0 to select the mutex.
4'd2	ptr2	Use ptr2 bit 8 concatenated with bits 3:0 to select the mutex.
4'd3	ptr3	Use ptr3 bit 8 concatenated with bits 3:0 of (ptr3 xor processIndex<3:0>) to select the mutex. See Operand A addressing modes table for details.
4'd4	ptr4	Use ptr4 bit 8 concatenated with bits 3:0 to select the mutex.
4'd5	userdef0	First of 10 user defined mutexes, available for microcode to use however it wants.
4'd6	userdef1	User defined mutex
4'd7	userdef2	User defined mutex
4'd8	userdef3	User defined mutex
4'd9	userdef4	User defined mutex
4'd10	userdef5	User defined mutex
4'd11	userdef6	User defined mutex
4'd12	userdef7	User defined mutex
4'd13	userdef8	User defined mutex
4'd14	userdef9	Last of 10 user defined mutexes
4'd15		reserved

5.5.14 Internal Encoding of Sleep Conditions

The sleep index field uses instruction bits plus parts of the thread state to select a particular hardware condition or mutex. Inside the DMA microengine, conditions and mutexes are treated almost the same. Conditions and mutexes resolve to a six-bit condition number that the thread selector can use to decide when to wake up a thread. The following table lists all the conditions that can cause a thread to sleep, and how they are encoded. The sleepCond register in the microengine (visible on R_SDmaSleepCondL and R_SDmaSleepCondH status registers) is a bit field whose bit numbers are defined by the Constant in this table.

Enum

DmaUeSleepCond

Constant	Mnemonic	Definition	(Controlled by)
6'h00	MUTEX0	First of 32 mutexes selected by ptr0-ptr4 value. A one in this bit means that the mutex is available; zero means that the mutex is unavailable.	uCode
6'h1F	MUTEX31	Last of 32 mutexes selected by ptr0-ptr4 value.	uCode
6'h20	MUTEX_USERDEF0	First of 10 mutexes selected by userdef0-userdef9	uCode
6'h29	MUTEX_USERDEF9	Last of 10 mutexes selected by userdef0-userdef9	uCode
6'h2A	MEMDONE_THR0	All memory transfers started by thread 0 have completed. A zero in this bit means that thread 0 has started a transfer in the DMA cache interface which hasn't completed. One means that all transfers started by this thread have finished.	HW
6'h2B	MEMDONE_THR1	see above	HW
6'h2C	MEMDONE_THR2	see above	HW
6'h2D	MEMDONE_THR3	see above	HW
6'h2E	MEMDONE_THR4	see above	HW
6'h2F	MEMDONE_THR5	see above	HW
6'h30	MEMDONE_THR6	see above	HW
6'h31	MEMDONE_THR7	see above	HW
6'h32	MEMDONE_THR8	see above	HW
6'h33	MEMDONE_THR9	All memory transfers started by thread 9 have completed.	HW
6'h34	RX0_AVAIL	A new packet is available in receive port 0. If this bit is one, a packet has arrived in the receive port and is ready to be processed. If zero, the microengine must wait for a packet to arrive.	HW
6'h35	RX1_AVAIL	A new packet is available in receive port 1	HW
6'h36	RX2_AVAIL	A new packet is available in receive port 2	HW
6'h37	RX_COPY_AVAIL	A new packet is available in the receive side of the copy port	HW
6'h38	TX0_AVAIL	Empty packet buffer is available in transmit port 0. If this bit is one, the transmit port is ready for the microengine to send a packet; if zero, the microengine must wait before sending a transmit packet.	HW
6'h39	TX1_AVAIL	Empty packet buffer is available in transmit port 1	HW
6'h3A	TX2_AVAIL	Empty packet buffer is available in transmit port 2	HW
6'h3B	TX_COPY_AVAIL	Empty packet buffer is available in the transmit side of the copy port	HW
6'h3C		reserved	
6'h3D	IO_THREAD_AWAKE	Used to awaken the I/O processing thread during I/O operations. Default value is 0.	HW
6'h3E	HALT	HALT is a sleep condition that is always false. If a thread sleeps on this condition, it will never wake up.	Constant
6'h3F	NONE	In an instruction, this value means "don't sleep." In the sleepCond field of a thread's state, it means "I'm not sleeping."	Constant
6'h00	FIRST_MUTEX	the first entries are hardware flags. Which is the first entry that is a mutex? Update if encoding table changes.	
6'h29	LAST_MUTEX	last entry that is a mutex	

5.5.15 Branch Field

Enum
DmaUeInstBranch
Attributes
-allowlc

Constant	Mnemonic	Definition	(NumWays)
3'b000	GOTO	Unconditional branch (default)	1
3'b001	B<11:8>	Command dispatch on B operand	16
3'b010	A<1:0>	dispatch on port number in A operand	4
3'b011	NZ	NextAddr<0> = ALU Z NextAddr<1> = ALU N	4
3'b100	N	ALU N	2
3'b101	Z	ALU Z	2
3'b110-3'b111		Reserved	

5.5.16 Dedicated Microinstruction Addresses

I/O space operations make address bits 19:16 available as the process index. (See ptr3 definition in Operand A addressing.)

Address bits 6:3 are ANDed with a 4-bit kernel-programmable mask to produce microinstruction address bits 3:0. (See PROG_IO register)

Microinstruction address bits 7:4 are 0 for I/O writes, 1 for I/O reads, and 2 for SPCL writes.

Defines

DMA_UINST_ADDR

Constant	Mnemonic	Definition
10'h00	PROG_IO_WRITE	For programmable I/O writes, execute microcode at this address plus the I/O write address bits 6:3
10'h10	PROG_IO_READ	For programmable I/O reads, execute microcode at this address plus the I/O write address bits 6:3
10'h20	PROG_IO_SPCL	For programmable SPCLs, execute microcode at this address plus the SPCL address bits 6:3
10'h30	DEFAULT_ENTRY_THR0	First instruction executed by thread 0
10'h31	DEFAULT_ENTRY_THR1	First instruction executed by thread 1
10'h39	DEFAULT_ENTRY_THR9	First instruction executed by thread 9

5.5.17 Miscellaenous Constant Definitions

Defines

DMA

Constant	Mnemonic	Definition
32'd4	PBUF_N	Number of packet buffers in a receive or transmit port. PBUF_N = 1 << PBUF_N_LOG_2
32'd2	PBUF_N_LOG_2	How many bits are required to encode the packet buffer number PBUF_N?
32'd3	PBUF_BUF_MASK	Bitmask used for selecting the buffer number in the low bits of the PBUF address. PBUF_MASK = PBUF_N - 1
32'd64	PBUF_WORDS	Number of words in a receive port packet buffer
32'd72	PBUF_BITS	Number of bits in packet buffer
32'd72	OPRF_BITS	Number of bits in receive port operand regfile.
32'd44	OPRF_WORDS	Number of words in a receive port operand regfile. 32 words of fabric switch control/staus registers + 3 regs * 4 packet buffers = 44. The three regs are pktHead, pktCtl, and pktTrail.
32'd5	N_OPERAND_PTRS	Number of pointers in operand A, B, and destination
32'd72	DMEM_BITS	Number of bits in microengine data memory
32'd1024	DMEM_WORDS	Number of words in microengine data memory. It's split into two halves, each DMEM_WORDS/2.
32'd4	DMEM_INTERLEAVE_BIT	Which bits of DMEM address determines interleaving of data between the four banks halves. The banks are interleaved on bits DMEM_INTERLEAVE_BIT and DMEM_INTERLEAVE_BIT+1.
10'h100	DMEM_PROCESS0	Address in DMEM of the first process descriptor
10'h020	DMEM_PROCESS_INCR	Add this to the DMEM address to find the next process. The address for process descriptor P would be DMEM_PROCESS0 + P * DMEM_PROCESS_INCR.
32'd64	UIM_BITS	Number of bits in microinstruction memory
32'd1024	UIM_WORDS	Number of words in microinstruction memory
32'd10	UIM_ADDR_BITS	Number of bits needed to specify an address in the microinstruction memory. 1<<UIM_ADDR_BITS = UIM_WORDS.
32'd10	N_THREADS	Number of threads in microengine
32'd2	MAX_TASKS_PER_THREAD	Number of cache interface operations per thread
32'd4	OUTSTANDING_READS	ICE9 only: Maximum number of outstanding reads from DMA to L2 Memory Bus
32'd4	OUTSTANDING_WRITES	ICE9 only: Maximum number of outstanding writes from DMA to L2 Memory Bus
32'd7	OUTSTANDING_READS_TWC	TWC9 only: Maximum number of outstanding reads from DMA to L2 Memory Bus
32'd7	OUTSTANDING_WRITES_TWC	TWC9 only: Maximum number of outstanding writes from DMA to L2 Memory Bus
32'd4	NUM_MEMOUT_SEQ	Number of MemOut address sequencers. There are four sequencers, one for each of: rxp0, rxp1, rxp2, and copy ports.
32'd4	NUM_MEMIN_SEQ	Number of MemOut address sequencers. There are four sequencers, one for each of: txp0, txp1, txp2, and copy ports.
8'hA0	RMB_IO_CORE0	address in memory read buffer (RMB) where I/O write data from core 0 is stored
8'hA8	RMB_IO_CORE1	addr in RMB where data from core 1 is stored
8'h08	RMB_IO_ADDR_INCR	distance between I/O data addresses. Use RMB_IO_CORE0 + N * RMB_IO_ADDR_INCR
8'hA0	WMB_IO_CORE0	address in memory write buffer (WMB) where I/O read data for core 0 is stored
8'hA8	WMB_IO_CORE1	addr in WMB where data for core 1 is stored
8'h08	WMB_IO_ADDR_INCR	distance between I/O data addresses. Use WMB_IO_CORE0 + N * WMB_IO_ADDR_INCR

5.5.18 DMA Thread Numbers

This table shows what tasks are assigned to the DMA microengine threads.
Defines
DMA_THR

Constant	Mnemonic	Definition
32'd0	RX0	Thread that services receive port 0
32'd1	RX1	Thread that services receive port 1
32'd2	RX2	Thread that services receive port 2
32'd3	COPY_RX	Thread that services the receive side of the copy port
32'd4	TX0	Thread that services transmit port 0
32'd5	TX1	Thread that services transmit port 1
32'd6	TX2	Thread that services transmit port 2
32'd7	COPY_TX	Thread that services the transmit side of the copy port
32'd8	QUEUE_MGR	Queue manager thread
32'd9	IO_ACCESS	Thread that handles I/O accesses from cores
32'd10	N_THREADS	Number of threads in microengine

5.5.19 DMA Port numbers

Enum
DmaPort

Constant	Mnemonic	Definition
3'd0	RX0	Receive port 0
3'd1	RX1	Receive port 1 control registers (read only)
3'd2	RX2	Receive port 2 control registers (read only)
3'd3	RXCOPY	Copy port memories, receive side
3'd4	TX0	Transmit port 0 control registers (write only)
3'd5	TX1	Transmit port 1 control registers (write only)
3'd6	TX2	Transmit port 2 control registers (write only)
3'd7	TXCOPY	Copy port memories, transmit side

5.5.20 DMA Queue numbers

These constants are chosen to match the order in the Common Control/Status (Kernel R/W) table. If that table is converted to a form that vspecs can read, then DmaQueue is redundant and should be removed.
Enum
DmaQueue

Constant	Mnemonic	Definition
4'd0	RX0	Receive port 0 queue
4'd1	RX1	Receive port 1 queue
4'd2	RX2	Receive port 2 queue
4'd3	RXCOPY	Copy port queue, receive side
4'd4	TX0	Transmit port 0 foreground queue
4'd5	TX1	Transmit port 1 foreground queue
4'd6	TX2	Transmit port 2 foreground queue
4'd7	TX0BG	Transmit port 0 background queue
4'd8	TX1BG	Transmit port 1 background queue
4'd9	TX2BG	Transmit port 2 background queue
4'd10	TXCOPY	Copy port memories, transmit side

5.5.21 DMA Internal Memory Addresses

Class
DmaInternalAddr
Attributes

Bit	Mnemonic	Type	Constant	Definition
w0[5:0]	mem	DmaInternalMem		The mem field tells which of the DMA's memories is selected
w0[15:6]	index			The index field tells the address in the selected memory. 10 bits wide.
w0[15:0]	allBits			for reading the whole structure as a single bit vector. Overlaps allowed.

5.5.22 DMA Internal Memory Addresses (Mem Field)

This table creates an encoding for every memory in the DMA engine. The encodings are used for several different purposes, including operand and destination selection and memory<=>cache transfers. These values are useful to circuit implementors but not to programmers.

It is important that the memories addressed by the cache interface are grouped together so that some number of low bits of the constant can distinguish them.

Enum

DmaInternalMem

Constant	Mnemonic	Definition
6'h0	NONE	no memory selected
6'h2	IMEM	Microengine instruction memory
6'h3	DMEM	Microengine data memory
6'h5	UE_REGS	registers in microengine, e.g. THREAD_SEL
6'h11	RX0_PKR	RX port 0 packet header/trailer registers
6'h15	RX1_PKR	RX port 1 packet header/trailer registers
6'h19	RX2_PKR	RX port 2 packet header/trailer registers
6'h1C	RXCOPY_PKR	RX copy port packet header/trailer registers
6'h21	TX0_PKR	TX port 0 packet header/trailer registers
6'h25	TX1_PKR	TX port 1 packet header/trailer registers
6'h29	TX2_PKR	TX port 2 packet header/trailer registers
6'h2C	TXCOPY_PKR	TX copy port packet header/trailer registers
6'h30	RX0_PBUF	RX port 0 packet buffers. NOTE: all packet buffers are 0x30 to 0x3F. Circuits that only refer to packet buffers don't need to store all 6 bits. They can just use values like RX2_PBUF - RX0_PBUF and store only 4 bits. Also, it's important that the first 8 packet buffers starting with RX0_PBUF are in thread order.
6'h31	RX1_PBUF	RX port 1 packet buffers
6'h32	RX2_PBUF	RX port 2 packet buffers
6'h33	RXCOPY_PBUF	copy port packet buffers (for reading)
6'h34	TX0_PBUF	TX port 0 packet buffers
6'h35	TX1_PBUF	TX port 1 packet buffers
6'h36	TX2_PBUF	TX port 2 packet buffers
6'h37	TXCOPY_PBUF	copy port packet buffers (for writing)
6'h38	RMB0	Read memory buffer 0, in copy port. In fact RMB0 and RMB1 are two adjacent regions in the same memory.
6'h39	RMB1	read memory buffer 1, in copy port
6'h3A	WMB0	write memory buffer 0, in copy port
6'h3B	WMB1	write memory buffer 1, in copy port

5.5.23 Receive Port Buffer State Machine

Enum

DmaRxpState

Constant	Mnemonic	Definition
2'b00	ST_SWRX	transfer from switch pending
2'b01	ST_WAITUE	tx from switch done. wait to enter ST_UE.
2'b11	ST_UE	selected for microengine operations
2'b10	ST_CA	cache operation pending

5.5.24 Receive Port CMUX Select Values

Enum

DmaRxpCmuxSel

Constant	Mnemonic	Definition
4'b0000	NONE	select nothing. cmux will output all zeroes.
4'b0100	RXP0	select data from receive port 0.
4'b0101	RXP1	select data from receive port 1.
4'b0110	RXP2	select data from receive port 2.
4'b0111	COPY	select data from copy port.
4'b0111	UNIT_SEL_MASK	bits 2,1,0 indicate which unit is selected.
4'b1000	ENABLE_ODD_WORD	bit 3=1 enables the odd word. bit 3=0 clears the odd word.

5.5.25 Transmit Port Buffer State Machine

Enum

DmaTxpState

Constant	Mnemonic	Definition
2'b00	IDLE	tx from switch done. wait to enter ST_UE.
2'b01	UE	selected for microengine operations
2'b11	CA	cache operation pending
2'b10	SWTX	transfer from switch pending

5.5.26 Transmit Port: Packet Builder State Machine

Enum

DmaTxpBldPktState

Constant	Mnemonic	Definition
3'b000	IDLE	ready to accept data from switch
3'b001	SEND_FORD1	sending header FORD
3'b101	SEND_FORD2	sending control FORD
3'b010	SEND_REGFILE	sending data from packet buffer
3'b011	SEND_TRAILER	sending trailer FORD

5.5.27 Copy Port Buffer State Machine

Enum

DmaCopyState

Constant	Mnemonic	Definition
3'h0	IDLE	waiting to enter UETX
3'h1	UETX	selected for microengine operations in Copy TX thread
3'h3	RDMEM	read operations pending in DmaCif
3'h7	RDY	waiting to enter UERX
3'h6	UERX	selected for microengine operations in Copy RX thread
3'h4	WRMEM	write operations pending in DmaCif

5.5.28 Copy Port: Read/Write Memory Buffer Address

Class

DmaCopyMbAddr

Attributes

Bit	Mnemonic	Type	Constant	Definition
w0[7:4]	thread			Thread number, 0-9. Also, I/O reads and writes use these buffers to store data being read or written, by setting thread to (10+core_number).
w0[3]	block			which of the thread's two cache blocks are accessed, 0 or 1
w0[2:0]	dwords			the low 3 bits select which doubleword within a cache block
w0[7:0]	allBits			for reading the whole field as one bit vector. Overlaps allowed.

5.5.29 Dma Cache Interface Task

used in ReadWriteQ, ReadWriteExtQ, OutstandingWriteTable

Class

DmaCifTask

Attributes

Bit	Mnemonic	Type	Constant	Definition
w0[3:0]	thread			microengine thread number
w0[7:4]	localTarget			which internal DMA unit will be accessed. Encoding is the low 4 bits of DmaInternalMem.
w0[12:8]	dwordsLeft			number of doublewords remaining to be transferred (0-19)
w0[20:13]	localAddr			address in local DMA memory, 8 bits
w0[53:21]	memAddr			address in main memory. There are 33 bits for Address<35:3>.
w0[56:54]	type			what command to send to CSW? In block read and write queues, only BRD or BWT will appear.
w0[57]	firstBlock32Byte			1=This is the first cache block transfer in a transfer that starts on a half-cache-block boundary. 0=any subsequent blocks
w0[58]	swapEvenOdd			is the memory address aligned to a doubleword or not? if 0, it is aligned. if 1, enable 32-bit swap.
w0[59]	valid			is this a valid task or just a No-op? 1=valid task. 0=no operation. ignore all the other bits in this task.
w0[63:0]	allBits			for reading the whole field as one bit vector. Overlaps allowed.

5.5.30 Dma Cache Interface: Memory Operation Type

These encodings are used on the ue_cif_TaskType_c5a bus.

Enum

DmaUeMemOpType

Constant	Mnemonic	Definition
4'd0	BWT	Start write operation from an internal DMA memory to the L2. The length of transfer comes from the payload length register in the port associated with this thread. (only threads 0-7)
4'd1	BRD	Start read operation from the L2 to an internal DMA memory. The length of transfer comes from the payload length register in the port associated with this thread. (only threads 0-7)
4'd2	BWT8	Same as BWT except the length is 8 bytes. The hardware will transfer a whole cache block, but the thread may be able to awaken sooner than if it asked for a 64 byte transfer.
4'd3	BRD8	Same as BRD except the length is 8 bytes. The hardware will transfer a whole cache block, but the thread may be able to awaken sooner than if it asked for a 64 byte transfer.
4'd4	BWT32	Same as BWT except the length is 32 bytes.
4'd5	BRD32	Same as BRD except the length is 32 bytes.
4'd6	BWT64	Same as BWT except the length is 64 bytes.
4'd7	BRD64	Same as BRD except the length is 64 bytes.
4'd8	BWT96	Same as BWT except the length is 96 bytes.
4'd9	BRD96	Same as BRD except the length is 96 bytes.
4'd10	BWT128	Same as BWT except the length is 128 bytes.
4'd11	BRD128	Same as BRD except the length is 128 bytes.
4'd12		Reserved
4'd13	IORD	Response to I/O read from a core. Drive Data only. This memory op does not increment the thread counter.
4'd14	SPCL	Response to a SPCL from a core. Drive DONE command onto CmdAddr bus. This memory op does not increment the thread counter.
4'd15	INTR	Send an interrupt. The bus stop number will be on alu_cif_MemAddr<15:12> and the unique id will be on alu_cif_MemAddr<11:0>. This memory op does not increment the thread counter.

5.5.31 Dma Cache Interface: Type of Task

These encodings are used for the type field in the Task data structures.

Enum

DmaCifTaskType

Constant	Mnemonic	Definition
3'b000	BWT	Start block write to coherence controller. Drive BWT on CmdAddr.
3'b001	BRD	Start block read. Drive RDS (????) command on CmdAddr, then wait for Data to arrive.
3'b010	PRBDONE	End of block read protocol. After data arrives, only if ????, send a PRBDONE on CmdAddr to notify COH that read is complete.
3'b011	IOWR	When WRIO arrives from a core, send RDIO on CmdAddr as a response.
3'b100	IORD	Response to I/O read from a core. Drive Data only.
3'b101	INTR	Send an interrupt to a processor. See "sendIntr" in the DmaUeInstMemOp field for more details.
3'b110	BRDR	Block read retry
3'b111	SPCL	SPCL command, a CmdAddr-only command that triggers a programmable I/O operation. When SPCL is in the StartIoQ it causes the microengine to execute an I/O operation. When SPCL is in the WriteQ it causes the DmaCif to send a DONE command back to the processor.

5.5.32 Dma Cache Interface: Numbering of Queues

Enum
DmaCifQueueNum

Constant	Mnemonic	Definition
4'd0	CRWQ	read/write queue
4'd2	CRWEXTQ	read/write extended queue
4'd4	CSPCLINTQ	queue fo SPCL responses and interrupts
4'd5	CRDIOQ	RDIO command queue (responses to WTIOs)
4'd6	CBRDRQ	block read retry queue
4'd7	DATARESPQ	data response queue
4'd8	CSKIDQ	1-deep skid buffer, holds a dequeued task during stall cycles
4'd9	DRDIOQ	RDIO data queue
4'd10	DWQ	data write queue
4'd11	DSKIDQ	1-deep skid buffer, holds a dequeued data task during stall cycles
4'd12	IOQ	StartIo queue for I/O reads
4'd15	NONE	no queue selected

5.5.33 Dma Cache Interface: Depth of Queues for ICE9

Defines
DMA_QUEUE_SIZE

Constant	Mnemonic	Definition
8'd20	CRWQ	read/write queue
8'd20	CRWEXTQ	read/write extended queue
8'd16	CSPCLINTQ	queue fo SPCL responses and interrupts
8'd6	CRDIOQ	RDIO command queue (responses to WTIOs)
8'd4	CBRDRQ	block read retry queue
8'd4	DATARESPQ	data response queue
8'd1	CSKIDQ	1-deep skid buffer, holds a dequeued task during stall cycles
8'd6	DRDIOQ	RDIO data queue
8'd4	DWQ	data write queue
8'd1	DSKIDQ	1-deep skid buffer, holds a dequeued data task during stall cycles
8'd12	IOQ	StartIo queue for I/O reads

5.5.34 Dma Cache Interface: Depth of Queues for TWC9

Defines
DMA_QUEUE_SIZE_TWC

Constant	Mnemonic	Definition
8'd20	CRWQ	read/write queue
8'd20	CRWEXTQ	read/write extended queue
8'd20	CSPCLINTQ	queue fo SPCL responses and interrupts
8'd10	CRDIOQ	RDIO command queue (responses to WTIOs)
8'd7	CBRDRQ	block read retry queue
8'd7	DATARESPQ	data response queue
8'd1	CSKIDQ	1-deep skid buffer, holds a dequeued task during stall cycles
8'd10	DRDIOQ	RDIO data queue
8'd7	DWQ	data write queue
8'd1	DSKIDQ	1-deep skid buffer, holds a dequeued data task during stall cycles
8'd20	IOQ	StartIo queue for I/O reads

5.5.35 Dma Cache Interface: Outstanding Read Table entry

Class
DmaCifOrtEntry
Attributes

Bit	Mnemonic	Type	Constant	Definition
w0[0]	valid			this table entry is valid
w0[1]	swapEvenOdd			1=use 32-bit alignment
w0[4:2]	align			Alignment information for transmit buffer. MemAddr<5:3> is stored here so that when we send the packet to the FSW, the TX port knows how to align the data.
w0[12:5]	localAddr			address in local DMA memory
w0[45:13]	memAddr			so that we know the address for BRDR and PRB-DONE. We know there is duplication between align and memAddr, but we're leaving it because we thing memAddr can be eliminated.
w0[49:46]	localTarget			Which internal DMA unit will be accessed? The encoding is the low 4 bits of DmaInternalMem.
w0[53:50]	thread			thread number, needed for thread accounting
w0[63:0]	allBits			for reading the whole field as one bit vector. Overlaps allowed.

5.5.36 Dma Cache Interface: Outstanding Write Table entry

The OWT data is encoded using the DmaCifTask data structure.

5.5.37 Dma Cache Interface: Block Read Retry Queue (BrdrQ) for ICE9

Class
DmaCifProtocolEntry

Bit	Mnemonic	Type	Constant	Definition
w0[4:0]	tid			Transaction id bits
w0[8:5]	dest			L2 bus stop number of the block that we will write to
w0[9]	valid			this table entry is valid
w0[63:0]	allBits			for reading the whole field as one bit vector. Overlaps allowed.

5.5.38 Dma Cache Interface: Block Read Retry Queue (BrdrQ) for TWC9

Class
DmaTwcCifProtocolEntry

Bit	Mnemonic	Product	Type	Constant	Definition
w0[5:0]	tid	TWC9A			Transaction id bits
w0[9:6]	dest	TWC9A			L2 bus stop number of the block that we will write to
w0[10]	valid	TWC9A			this table entry is valid
w0[63:0]	allBits	TWC9A			for reading the whole field as one bit vector. Overlaps allowed.

5.5.39 Dma Cache Interface: Command RDIO Queue (CrdioQ)

This queue is encoded with DmaCifProtocolEntry.

5.5.40 Dma Cache Interface: SPCL/INT Queue (CSpclIntQ) for ICE9

Class
DmaCifSpclIntEntry

Bit	Mnemonic	Type	Constant	Definition
w0[4:0]	tid			Transaction id bits (for SPCL only)
w0[11:0]	intReason			Interrupt reason (for INT only). Overlaps tid.
w0[15:12]	dest			L2 bus stop number of the block that we will write to
w0[16]	isSpcl			which type of command is this? 1=SPCL, 0=INT
w0[20:17]	thread			Thread number (for INT only)
w0[21]	valid			this table entry is valid
w0[63:0]	allBits			for reading the whole field as one bit vector. Overlaps allowed.

5.5.41 Dma Cache Interface: SPCL/INT Queue (CSpclIntQ) for TWC9

Class
DmaTwcCifSpclIntEntry

Bit	Mnemonic	Product	Type	Constant	Definition
w0[5:0]	tid	TWC9A			Transaction id bits (for SPCL only)
w0[11:0]	intReason	TWC9A			Interrupt reason (for INT only). Overlaps tid.
w0[15:12]	dest	TWC9A			L2 bus stop number of the block that we will write to
w0[16]	isSpcl	TWC9A			which type of command is this? 1=SPCL, 0=INT
w0[20:17]	thread	TWC9A			Thread number (for INT only)
w0[21]	valid	TWC9A			this table entry is valid

5.5.42 Dma Cache Interface: Data Response Queue (DataRspQ)

This queue is encoded with DmaCifProtocolEntry.

5.5.43 Dma Cache Interface: Data Write Queue (DWQ)

This queue is encoded with DmaCifProtocolEntry.

5.5.44 Dma Cache Interface: I/O Read Queue (DRDIOQ)

This queue is encoded with DmaCifProtocolEntry.

5.5.45 Dma Cache Interface: StartIoQ for ICE9

Class

DmaCifStartIoEntry

Attributes

Bit	Mnemonic	Type	Constant	Definition
d0[1:0]	type	DmaCifStartIoType		RDIO or WTIO or SPCL
d0[34:2]	ioAddr			33 bits corresponding to csw_dma_Addr<35:3>
d0[39:35]	tid			L2 transaction id for this I/O operation
d0[43:40]	origin			bus stop number of originator
d0[63:0]	allBits			for reading all bits at once. Overlaps allowed.

5.5.46 Dma Cache Interface: StartIoQ for TWC9

Class

DmaTwcCifStartIoEntry

Attributes

Bit	Mnemonic	Product	Type	Constant	Definition
d0[1:0]	type	TWC9A	DmaCifStartIoType		RDIO or WTIO or SPCL
d0[34:2]	ioAddr	TWC9A			33 bits corresponding to csw_dma_Addr<35:3>
d0[40:35]	tid	TWC9A			L2 transaction id for this I/O operation
d0[44:41]	origin	TWC9A			bus stop number of originator
d0[63:0]	allBits	TWC9A			for reading all bits at once. Overlaps allowed.

5.5.47 Dma Cache Interface: StartIoType

These encodings are used for the type field in the StartIo data structure.

Enum

DmaCifStartIoType

Constant	Mnemonic	Definition
2'b00	RDIO	I/O operation is a read
2'b01	WTIO	I/O operation is a write
2'b10	SPCL	I/O operation is a special (one way message from core to DMA)
2'b11		reserved

5.5.48 Dma Cache Interface: Address memory entry

Class

DmaCifAdmEntry

Attributes

Bit	Mnemonic	Type	Constant	Definition
d0[35:0]	memAddr			address in main memory
d0[40:36]	len			number of doublewords to transfer
d0[63:0]	allBits			for reading the whole field as one bit vector. Overlaps allowed.

5.5.49 Dma Cache Interface: MemOut Address Sequencer States

Enum

DmaCifMoaState

Constant	Mnemonic	Definition
3'b000	IDLE	wait for start signal
3'b001	READ01	read doublewords 0 and 1
3'b010	READ23	read doublewords 2 and 3
3'b011	READ45	read doublewords 4 and 5
3'b100	READ67	read doublewords 6 and 7

5.5.50 Dma Cache Interface: MemIn Address Sequencer States

Enum

DmaCifMiaState

Constant	Mnemonic	Definition
3'b000	IDLE	wait for start signal
3'b001	WRITE01	write doublewords 0 and 1
3'b010	WRITE23	write doublewords 2 and 3
3'b011	WRITE45	write doublewords 4 and 5
3'b100	WRITE67	write doublewords 6 and 7

5.5.51 Internal Encodings for Microengine Operands

These values are used within the microengine on signals ue_xxx_OpaAddr_c3a, ue_xxx_OpbAddr_c3a, and ue_xxx_ResultAddr_c5a. Because many of the things the microengine can address are accessible from I/O, we're using I/O addresses even for some of the things that are internal.

Defines

DMA_OP_ENC

Constant	Mnemonic	Definition
24'h321300	PTR0	ptr0 of the current thread
24'h321301	PTR1	ptr1 of the current thread
24'h321302	PTR2	ptr2 of the current thread
24'h321303	PTR3	ptr3 of the current thread
24'h321304	PTR4	ptr4 of the current thread
24'h321310	IO_ADDR	ioAddr register
24'h321311	IO_DATA	ioData register
24'h321312	SPCL_DATA	spclData register

5.5.52 I/O Region Type (DmaIoRegionType)

This data type describes regions of I/O addresses in the table above.

Enum

DmaIoRegionType

Constant	Mnemonic	Definition
3'b101	FIXED_RW_OPA	Region is readable and writable by fixed I/O. Reads use operand A.
3'b011	FIXED_RW_OPB	Region is readable and writable by fixed I/O. Reads use operand B.
3'b111	NONE	not a valid region type

5.5.53 External I/O Addresses

Assume the DMA engine I/O space starts at DMA_IO_BASE. Everything else is specified as an offset relative to DMA_IO_BASE.

Defines

DMA_IO

Constant	Mnemonic	Definition
36'hE_8100_0000	BASE	Start of DMA engine's I/O space
36'hE_843F_FFFF	END	End of DMA engine's I/O space
24'h010000	PAGE_SIZE	These addresses are calculated based on a page size of 64kb = 0x10000 bytes. As of 5/16/2005 that was our best guess.
24'h320000		reserved
24'h321000	FIRST_UE_REG	Address of first DMA register whose value lives in the microengine module DmaUe
24'h321300		Reserved for internal encodings. See the table DMA_OP_ENC for details.

5.6 Registers Accessible by RDIO/WTIO from Processors

5.6.1 DMA Instruction Memory (IMEM)

Every location in the DMA instruction memory is I/O accessible. At node initialization time, every location must be initialized to a known value, to ensure repeatable results and to avoid false detection of ECC errors. The IMEM may only be accessed when every DMA thread is disabled (see R_DmaThreadSel).

Register

R_DmaImem[1023:0]

Address

0xE_8131_0000-0xE_8131_1FFF (Add 0x8 per entry)

Attributes

-kernel

Bit	Mnemonic	Access	Reset	Definition
63:0	Instr	RW	X	Allows read/write access to one word of IMEM.

5.6.2 DMA Data Memory (DMEM)

Every location in the DMA data memory is I/O accessible. At node initialization time, every location must be initialized to a known value, to ensure repeatable results and to avoid false detection of ECC errors. Usually the processors will not access Dmem while the microengine is running, but it is perfectly legal to do so.

Register

R_DmaDmem[1023:0]

Address

0xE_8130_0000-0xE_8130_1FFF (Add 0x8 per entry)

Attributes

-kernel

Bit	Mnemonic	Access	Reset	Definition
63:0	Data	RW	X	Allows read/write access to one word of DMEM.

5.6.3 DMA Thread Select Register

Register

R_DmaThreadSel

Attributes

-kernel

Address

0xE_8132_1100

Bit	Mnemonic	Access	Reset	Definition
9:0	threadEnable	RWS	0	The thread enable bits allow external software to control which threads execute and which do not. When the bit corresponding to a thread is 1, the thread is allowed to issue instructions, subject to the countdown behavior. When 0, the thread may not execute any instructions. The threadEnable bit corresponding to the I/O thread is ignored because the I/O thread cannot be disabled.
31:16	countdown	RW	0	This 16-bit counter allows software to ask the DMA engine to execute N instructions and then halt. When countdownHalt=1, the counter decrements as each microinstruction is issued, but when it reaches zero, all threads (except for the I/O thread) stop issuing instructions until software intervenes.
32	countdownHalt	RW	0	When 1, enable countdown-and-halt behavior described above. When 0, disable countdown-and-halt behavior.

Cautionary Note: ThreadEnable bits must be used with caution: any thread can take a mutex flag which may be needed by the I/O thread in order to service a read, write, or spcl request (that is, requests to DmaAppIface0 or DmaAppIface1). If a processor issues such a request while a stopped thread is holding such a mutex, the processor will be hung and must be reset to recover.

In current microcode (as of March 2006), only writes of eventQRdSize depend on a mutex.

5.6.4 DMA Thread Pointer Registers

This table describes the thread pointer registers. There are 10 in all, one for each DMA microengine thread.

Register

R_DmaThreadPtr[9:0]

Address

0xE_8132_1000-0xE_8132_104F (Add 0x8 per entry)

Attributes

-kernel

Bit	Mnemonic	Access	Reset	Definition
d0[9:0]	ptr0	RW	0	Pointer into dmem
d0[19:10]	ptr1	RW	0	Pointer into dmem
d0[29:20]	ptr2	RW	0	Pointer into dmem
d0[39:30]	ptr3	RW	0	Pointer into dmem
d0[49:40]	ptr4	RW	0	Pointer into dmem

5.6.5 DMA Thread Program Counter Registers

This table describes the thread PC registers. There are 9 in all, one for each DMA microengine thread except for the I/O thread #9. The I/O thread has internal registers for pc and sleepCond, but they are not visible to software because the act of reading or writing an I/O register affects the I/O thread's values.

Register

R_DmaThreadPc[8:0]

Address

0xE_8132_1080-0xE_8132_10C7 (Add 0x8 per entry)

Attributes

-kernel

Bit	Mnemonic	Access	Reset	Type	Definition
9:0	pc	RW	0		Program counter for the thread. The pc tells what address in instruction memory to read.
15:10	sleepCond	RW	NONE	DmaUeSleepCond	This field indicates whether the thread is waiting for a condition to become true. If sleepCond is set to DmaUeSleepCond_NONE, the thread is NOT waiting for any condition; otherwise the field encodes which condition it is waiting for.

5.6.6 DMA Programmable I/O Control Register

Register
R_DmaProgIo
Attributes
-kernel
Address
0xE_8132_1108

Bit	Mnemonic	Access	Type	Reset	Definition
3:0	ioAddrMask	RW		0xf	For programmable I/O operations, the ioAddrMask bits are ANDed with the I/O address bits when generating the microinstruction address to execute.

5.6.7 DMA Application Interface Region 0

This is an address range in which loads and stores causes the DMA to execute microcode.
Register
R_DmaAppIface0[0x1FFFF:0]
Address
0xE_8110_0000-0xE_811F_FFF8 (Add 0x8 per entry)
Attributes
-noregtest -kernel

Bit	Mnemonic	Access	Type	Reset	Definition
63:0	Data	RW		X	Programmable I/O region 0. A load or store to this address range in a processor causes a RDIO and WTIO command on the CSW, which triggers a sequences of microcode in the DMA engine. WTIO to address X causes the microengine to execute instructions starting at IMEM address DMA_UINST_ADDR_PROG_IO_WRITE + (X[6:3] & ioAddrMask). RDIO from address X causes the I/O thread in the microengine to execute instructions starting at IMEM address DMA_UINST_ADDR_PROG_IO_READ + (X[6:3] & ioAddrMask).

5.6.8 DMA Application Interface Region 1

This is an address range in which stores cause the DMA to execute microcode.
Register
R_DmaAppIface1[0x1FFFF:0]
Address
0xE_BE20_0000-0xE_BE2F_FFF8 (Add 0x8 per entry)
Attributes
-noregtest -kernel

Bit	Mnemonic	Access	Type	Reset	Definition
63:0	Data	W		X	Programmable I/O region 1. A store to this address range in a processor causes a SPCL commands on the CSW, which triggers a sequences of microcode in the DMA engine. SPCL to address X causes the micro-engine to execute instructions starting at IMEM address DMA_UINST_ADDR_PROG_IO_SPCL + (X[6:3] & ioAddrMask).

5.7 Registers Accessible by Serial Configuration Bus

The DMA block has registers accessible by RDIO/WTIO and others accessible by the SCB. All SCB registers have the prefix "R_SDma" to indicate that they are on the SCB.

5.7.0.1 Block Reset Register

This register allows the RX/TX ports of the DMA to be reset individually. Each port has an active-high signal which forces everything back to its reset state. After the DMA block is reset, the ports remain in reset until software initializes the DMA and decides to allow packets to flow. This ensures that an unconfigured DMA cannot cause the fabric to back up.

Register

R_SDmaBlockReset

Attributes

-kernel

Address

0xE_0100_0000

Bit	Mnemonic	Access	Reset	Type	Definition
31:6					Reserved
5:3	TxReset	RW	7		One bit per transmit port. Bit 3+N affects TX port N. When reset is high, all state in the transmit port is cleared. The SoP, EoP, and DatVal signals to the fabric switch are held low. TxpN_ue_BufAvail_c1a is deasserted so that the microengine believes that all packet buffers are full.
2:0	RxReset	RW	7		One bit per receive port. Bits 2:0 affect RX2,1,0. When reset is high, all state in receive port is cleared. Dma_fsw_RdyN_s1a is asserted so that any incoming fabric packets are accepted and dropped. RxpN_ue_BufAvail_c1a is deasserted so that the micro-engine believes that no packets have arrived.

5.7.0.2 ECC Mode Register

Register

R_SDmaEccMode

Attributes

-kernel

Address

0xE_0100_0004

Bit	Mnemonic	Access	Reset	Type	Definition
31:7					Reserved
6	CifCorrEna	RW	1		Enable ECC correction in CIF. This logic is only needed when the microengine does a BRD from a memory address with bit 2 set (32-bit realignment). Bug2396: When CifCorrEna is **off** and the microengine does a BRD from a memory address with bit 2 set, the ECC written into the DMA's internal memory (TX or COPY port packet buffer) is incorrectly forced to zero. Data with corrupted ECC may reach the FSW or main memory when the packet is sent. The safest workaround is to always leave CifCorrEna on.
5	ImemCorrEna	RW	1		Enable ECC correction during Imem reads
4	DmemCorrEna	RW	1		Enable ECC correction during Dmem reads
3	CopyCorrEna	RW	1		Enable ECC correction when the Copy port reads a memory and places data onto the Operand B bus
2:0	RxpCorrEna	RW	7		Enable ECC correction when the RX port reads memory and places data onto the Operand B bus

5.7.0.3 ALU Merge Operation Control Registers (added in Twice9)

Register
R_SDmaMergeOpHi[3:0]
Attributes
-kernel -noregtest
Address
0xE_0100_0020-0xE_0100_002C

Bit	Mnemonic	Access	Reset	Type	Product	Definition
31:0	Hi	RW	0		TWC9A	These four registers control the operation of the DMA ALU operation Merge0, Merge1, Merge2, and Merge3. R_SDmaMergeOpHi[N] controls bits 63:32 of the MergeN result, while R_SDmaMergeOpLo[N] controls bits 31:0 of the MergeN result. See 5.2.12.4 for details.

Register
R_SDmaMergeOpLo[3:0]
Attributes
-kernel -noregtest
Address
0xE_0100_0030-0xE_0100_003C

Bit	Mnemonic	Access	Reset	Type	Product	Definition
31:0	Lo	RW	0		TWC9A	These four registers control the operation of the DMA ALU operation Merge0, Merge1, Merge2, and Merge3. R_SDmaMergeOpHi[N] controls bits 63:32 of the MergeN result, while R_SDmaMergeOpLo[N] controls bits 31:0 of the MergeN result. See 5.2.12.4 for details.

5.7.0.4 Force Error Register

This register causes the circuit to intentionally produce specific errors. This will help us to test error detection logic and error handling software.

Register
R_SDmaForceErr
Attributes
-kernel
Address
0xE_0100_0008

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved
7:6	DmemFlipMemBits	RW	0		These bits are XORed with bits 1 and 0 of every word of data being written to the data memory. If a corrupted data is read from Dmem, ECC correction logic in the Dmem (if enabled) will detect the error and set a bit in R_SDmaIntCause.
5:4	ImemFlipMemBits	RW	0		These bits are XORed with bits 1 and 0 of every word of data being written to the instruction memory. If a corrupted data is read from Imem, ECC correction logic in the Imem (if enabled) will detect the error and set a bit in R_SDmaIntCause.
3:2	CopyFlipMemBits	RW	0		These bits are XORed with bits 1 and 0 of every word of data being written to the copy port packet buffer and read/write memory buffer. Corrupted data in the packet buffer will be sent out the CSW to another block. Corrupted data in the read/write memory buffer will be corrected if the microengine reads it, but if it written back to CSW it will not be corrected by DMA at all.
1:0	TxFlipMemBits	RW	0		These bits are XORed with bits 1 and 0 of every word of data being written to the packet buffer of every transmit port. This field allows software to intentionally corrupt the data that is sent out the TX port to the fabric switch, to test the ECC correction logic in the fabric switch.

5.7.0.5 Microengine Status Registers

Register
R_SDmaUeStatus1
Address
0xE_0100_0108

Bit	Mnemonic	Access	Reset	Type	Definition
31:4					Reserved
3:0	PrevThread	R	0		Which thread ran last (0-9)

Register
R_SDmaUeSleepCondsL
Address
0xE_0100_0100

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	SleepCondsL	R	X		Lower 32 bits of the SleepCond vector in the microengine. For each bit, 1 means that the condition is “available” or “ready”. 0 means that any thread waiting for that condition would continue to wait. The bit numbers of the SleepCond vector are defined by the enum DmaUeSleepCond. Example: Does thread 3 have a memory operation outstanding in the DMA cache interface? The DmaUeSleepCond table has a row called MEMDONE_THR3 whose value is 0x2D. So you’d read SleepCondsH and SleepCondL, concatenate them into a 64-bit vector, and look at bit number 0x2D. If that bit is zero, thread 3 has a memory operation outstanding.

Register
R_SDmaUeSleepCondsH
Address
0xE_0100_0104

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	SleepCondsH	R	X		Upper 32 bits of the SleepCond vector in the microengine. See SleepCondL for details.

5.7.0.6 Cache Interface Status Registers

Register

R_SDmaCifStatus1

Address

0xE_0100_0110

Bit	Mnemonic	Access	Reset	Type	Definition
31:24					Reserved
23:16	RefCntZero	R	0xFF		Reads the 8 RefCntZero signals that go from the cache interface to the various ports. Use the DmaPort enum to decide which bit represents which bit, e.g. bit 8+DmaPort::TX0 represents cif_txp0_RefCntZero_c5a.
15:12	WriteTidBusy	R	0		A copy of the TidBusy wires for the 4 DMA write TIDs
11:8	ReadTidBusy	R	0		A copy of the TidBusy wires for the 4 DMA read TIDs
7:4	OwtValid	R	0		Valid bits of the outstanding write table. If bit 4+X is set, the DMA has an outstanding write on DMA write tid X.
3:0	OrtValid	R	0		Valid bits of the outstanding read table. If bit X is set, the DMA has an outstanding read on DMA read tid X.

Register

R_SDmaCifStatus2

Address

0xE_0100_0114

Bit	Mnemonic	Access	Reset	Type	Definition
31:16	OwtThread	R	X		Four fields of four bits each. Bits (19+4*X to 16+4*X) are the thread number of the Outstanding Write Table entry X.
15:0	OrtThread	R	X		Four fields of four bits each. Bits (3+4*X to 4*X) are the thread number of the Outstanding Read Table entry X.

5.7.0.7 Rx/Tx Port Status Registers

There are three port status registers, on for each RX and TX port. R_SDmaPortStatus[N] gives the status of RX port N and TX port N.

Register

R_SDmaPortStatus[2:0]

Address

0xE_0100_0120-0xE_0100_0128

Bit	Mnemonic	Access	Reset	Type	Definition
31:26					Reserved
25:24	TxWhichBuf	R	0		In the transmit port, which packet buffer is the microengine working on?
23:16	TxBufState	R	X		Read the packet buffer state. This field contains four bit fields of 2 bits each. Bits (17+2*M to 16+2*M) gives the state of packet buffer M. The 2-bit fields are of type DmaTxpState.
15:10					Reserved
9:8	RxWhichBuf	R	0		In the receive port, which packet buffer is the microengine working on?
7:0	RxBufState	R	X		Read the packet buffer state. This field contains four bit fields of 2 bits each. Bits (1+2*M to 2*M) gives the state of packet buffer M. The 2-bit fields are of type DmaRxpState.

5.7.0.8 Copy Port Status Register

Register

R_SDmaCopyPortStatus

Address

0xE_0100_0130

Bit	Mnemonic	Access	Reset	Type	Definition
31:16					Reserved
15:14	CopyTxWhichBuf	R	0		In the copy port, which packet buffer is the DMA_THR_COPY_TX thread of the microengine working on?
13:12	CopyRxWhichBuf	R	0		In the copy port, which packet buffer is the DMA_THR_COPY_RX thread of the microengine working on?
11:0	CopyBufState	R	X		Read the packet buffer state. This field contains four bit fields of 2 bits each. Bits 3*M gives the state of packet buffer M. The 2-bit fields are of type DmaRxpState.

5.7.0.9 Interrupt Cause Register

The interrupt cause register contains flags which are set when an event occurs, and cleared by software by writing a 1 to that bit.

Note on ECC correction and interrupt bits: Assuming correction is enabled, if a single bit error is detected, the data is corrected and the Sbe interrupt cause bit is set. If a double bit error is detected, both the Dbe interrupt cause bit and the Sbe interrupt cause bit are set, and the bad data will not be modified.

Register

R_SDmaIntCause

Address

0xE_0100_0200

Attributes

-kernel

Bit	Mnemonic	Access	Reset	Type	Definition
31	Intr	R	0		This bit is 1 when any bit in the expression R_SDmaIntCause[30:0] & R_SDmaIntMask[30:0] is set. It becomes the primary output dma_xxx_Int_ca.
30:15					Reserved.
14	CifDbe	RW1C	0		Cache Interface Double Bit Error. A double bit error has been detected in data read from the CSW. ECC correction/detection only occurs in the CIF if a block read is performed with address bit 2 equal to 1. If address bit 2 is 0, the CIF does not check ECC at all, and the data goes straight to the TX or copy port.
13	ImemDbe	RW1C	0		Imem Double Bit Error. A double bit error has been detected in data read from the Instruction Memory.
12	DmemDbe	RW1C	0		Dmem Double Bit Error. A double bit error has been detected in data read from the Data Memory.
11	CopyDbe	RW1C	0		Copy Port Double Bit Error. A double bit error has been detected in data read from the packet buffer or read/write memory buffer in the copy port. ECC correction/detection occurs if the microengine reads a corrupted data ford in the packet buffer or read/write memory buffer. But if the corrupted ford is written straight back to the CSW, no correction/detection occurs in DMA.
10:8	RxpDbe	RW1C	0		Receive Port Double Bit Error. Bit 8+N describes errors from RX port N. A double bit error has been detected in data read from the receive port packet buffer or the receive port operand memory. ECC correction/detection occurs if the microengine reads a corrupted data ford that came from the fabric switch. But if the packet is written straight to memory with a BRD, no correction/detection occurs in DMA.
7					Reserved
6	CifSbe	RW1C	0		Cache Interface Single Bit Error. A single bit error has been corrected in data coming from the CSW. See note in CifDbe description for when ECC correction occurs.
5	ImemSbe	RW1C	0		Imem Single Bit Error. A single bit error has been corrected in data read from the Instruction Memory.
4	DmemSbe	RW1C	0		Dmem Single Bit Error. A single bit error has been corrected in data read from the Data Memory.
3	CopySbe	RW1C	0		Copy Port Single Bit Error. A single bit error has been corrected in data read from the copy port packet buffer or read/write memory buffer. See note in CopyDbe description for when ECC correction occurs.
2:0	RxpSbe	RW1C	0		Receive Port Single Bit Error. A single bit error has been corrected in data read from the receive port packet buffer or operand regfile. Bit 0+N describes errors from RX port N. See note in RxpDbe description for when ECC cor-

5.7.0.10 Interrupt Mask Register

An interrupt mask register allows software to control which kinds of interrupts will cause the DMA's slow interrupt line to be asserted. Let's imagine that only double bit errors are of interest; software would write ones in R_SDmaIntMask for the bits corresponding to the double bit error interrupt causes in R_SDmaIntCause. Then, if any double bit error occurs, R_SDmaIntCause bit 31 would go up and the slow interrupt line would be asserted. If any other kind of error occurs, the R_SDmaIntCause bit would still go up, but bit 31 and the slow interrupt line would not be affected.

Register
R_SDmaIntMask
Address
0xE_0100_0204
Attributes
-kernel

Bit	Mnemonic	Access	Reset	Type	Definition
31					Reserved
30:0	IntMask	RW	0		If the corresponding interrupt cause bit is ever set, assert the interrupt.

5.8 SCB Performance Events

The following events are trackable by SCB statistical event counting.

Enum

DmaScbEvent

Attributes

-descfunc

Constant	Mnemonic	Definition
8'h00	CYCLES	Count every cycle. Drive 1 always.
8'h01	ECMD_ADDR_REQ	Request CSW command bus, evenbound
8'h02	OCMD_ADDR_REQ	Request CSW command bus, oddbound
8'h03	CMD_ADDR_GNT	Granted command bus, either direction
8'h04	CMD_ADDR_VALID	CSW command arrived at DMA
8'h05	EDATA_REQ	Request CSW data bus, evenbound
8'h06	ODATA_REQ	Request CSW data bus, oddbound
8'h07	DATA_GNT	Granted data bus, either direction
8'h08	DATA_VALID	CSW data arrived at DMA
8'h09	READ_MISS	cif_csr_ReadMiss_ca: Block reads that missed in L2 cache
8'h0A	READ_HIT	cif_csr_ReadHit_ca: Block reads that hit in L2 cache
8'h0B	WRITE_MISS	cif_csr_WriteMiss_ca: Block writes that missed in L2 cache
8'h0C	WRITE_HIT	cif_csr_WriteHit_ca: Block writes that hit in L2 cache
8'h0D-8'h1F		Reserved
8'h20	COPY_MEMIN_PBUF	cif_copy_MemInPbufSel_c4a: Cache blocks copied from memory to copy port
8'h21	COPY_MEMIN_RWMB	cif_copy_MemInRmbSel_c4a: Cache blocks copied from memory to r/w mem buffer
8'h22	COPY_MEMOUT_PBUF	cif_copy_MemOutPbufSel_c2a: Cache blocks copied from copy port to memory

8'h23	COPY_MEMOUT_RWMB	cif_copy_MemOutWmbSel_c2a: Cache blocks copied from copy port to memory
8'h24	TXP0_MEMIN	cif_txp_MemInTxp0Sel_c4a: Cache blocks copied from memory into TX port 0
8'h25	TXP1_MEMIN	cif_txp_MemInTxp1Sel_c4a: Cache blocks copied from memory into TX port 1
8'h26	TXP2_MEMIN	cif_txp_MemInTxp2Sel_c4a: Cache blocks copied from memory into TX port 2
8'h27	RXP0_MEMIN	cif_rxp_MemOutRxp0Sel_c2a: Cache blocks copied from RX port 0 to memory
8'h28	RXP1_MEMIN	cif_rxp_MemOutRxp1Sel_c2a: Cache blocks copied from RX port 1 to memory
8'h29	RXP2_MEMIN	cif_rxp_MemOutRxp2Sel_c2a: Cache blocks copied from RX port 2 to memory
8'h2A-8'h3F		Reserved
8'h40	UE_INSTR_VALID	ue_xxx_DbgValid_c2a: Instructions executed in microengine
8'h41	START_IO	cif_ue_StartIo_c1a: I/O reads, writes, and SP-CLs received by DMA.
8'h42	TASK_START	ue_cif_TaskStart_c5a: CSW operations started by the microengine.
8'h43	COPY_PORT_PKTS	ue_copy_TxThreadDone_c5a: Packets transferred out of the copy port.
8'h44-FF		Reserved.

5.9 Internal Data Formats and States

The data formats for some internal buses are documented here in the spec to help the SystemC and Verilog models stay in sync with each other. The only people who would care about these formats are the SystemC and Verilog authors. Everyone else can safely ignore this section.

5.9.1 Encoding of Buses between DmaCsr and DmaUe

5.9.1.1 CsrUeStat - For csr_ue_Stat_ca bus

Class

CsrUeStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:3]	U0		Unused. Drive 0.
d0[2]	EnableEcc		Enable ECC correction on Imem
d0[1:0]	FlipMemBits		XOR these bits with Imem data before writing.

5.9.1.2 UeCsrStat - For csr_ue_Stat_ca bus

Class

UeCsrStat

Bit	Mnemonic	Type	Definition
d1[63:32]	SleepCondsH		Connect to m_SleepCond_c2a[63:32]
d1[31:0]	SleepCondsL		Connect to m_SleepCond_c2a[31:0]
d0[63:6]	U0		Unused. Drive 0.
d0[5:2]	PrevThread		Which thread ran last (0-9)
d0[1]	DoubleBitErr		ECC corrector detected a double bit ECC error while reading instruction memory.
d0[0]	SingleBitErr		ECC corrector detected a single bit ECC error while reading instruction memory.

5.9.2 Encoding of Buses between DmaCsr and DmaCif

5.9.2.1 CsrCifStat - For csr_cif_Stat_ca bus

Class

CsrCifStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:3]	U0		Unused. Drive 0.
d0[2]	EnableEcc		Enable ECC correction
d0[1:0]	FlipMemBits		XOR these bits with the output of the ECC generator for cif_xxx_MemOutDw1_c4a during 32-bit realignment.

5.9.2.2 CifCsrStat - For csr_cif_Stat_ca bus

Class

CifCsrStat

Bit	Mnemonic	Type	Definition
d1[63:15]	U1		Unused. Drive 0.
d1[14:12]	DataArbCtr		3-bit arbitration counter for data queue selection
d1[11:10]	U1b		Unused. Drive 0.
d1[9:8]	CmdArbCtr		2-bit arbitration counter for command queue selection
d1[7:4]	DataSelQueue	DmaCifQueueNum	Which data queue was selected to go onto the dma_csw data bus?
d1[3:0]	CmdSelQueue	DmaCifQueueNum	Which command queue was selected to go onto the dma_csw command bus?
d0[63:48]	OwtThread		Provide OwtThread in R_SDmaCifStatus2
d0[47:32]	OrtThread		Provide OrtThread in R_SDmaCifStatus2
d0[31:26]			Reserved
d0[25]	DoubleBitErr		ECC corrector detected a double bit ECC error during 32-bit realignment of data from the CSW.
d0[24]	SingleBitErr		ECC corrector detected a single bit ECC error during 32-bit realignment of data from the CSW.
d0[23:16]	RefCntZero		Provide RefCntZero in R_SDmaCifStatus2
d0[15:12]	WriteTidBusy		Provide WriteTidBusy in R_SDmaCifStatus1
d0[11:8]	ReadTidBusy		Provide ReadTidBusy in R_SDmaCifStatus1
d0[7:4]	OwtValid		Connect to OWT valid bits
d0[3:0]	OrtValid		Connect to ORT valid bits

5.9.3 Encoding of Buses between DmaCsr and DmaDmem

5.9.3.1 CsrDmemStat - For csr_dmem_Stat_ca bus

Class

CsrDmemStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:3]	U0		Unused. Drive 0.
d0[2]	EnableEcc		Enable ECC correction
d0[1:0]	FlipMemBits		XOR these bits with Dmem data before writing.

5.9.3.2 DmemCsrStat - For csr_dmem_Stat_ca bus

Class
DmemCsrStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:2]	U0		Unused. Drive 0.
d0[1]	DoubleBitErr		ECC corrector detected a double bit ECC error while reading data memory.
d0[0]	SingleBitErr		ECC corrector detected a single bit ECC error while reading data memory.

5.9.4 Encoding of Buses between DmaCsr and DmaTxp

5.9.4.1 CsrTxpStat - For csr_txp_Stat_ca bus

Class
CsrTxpStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:2]	U0		Unused. Drive 0.
d0[1:0]	FlipMemBits		XOR these bits with ALU result data before writing to packet buffer or operand register file.

5.9.4.2 TxpCsrStat - For csr_txp_Stat_ca bus

Class
TxpCsrStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:10]	U0		Unused. Drive 0.
d0[9:8]	TxWhichBuf		Provide TxWhichBuf in R_SDmaPortStatus[X]
d0[7:0]	TxBufState		Provide TxBufState in R_SDmaPortStatus[X]

5.9.5 Encoding of Buses between DmaCsr and DmaRxp

5.9.5.1 CsrRxpStat - For csr_rxp_Stat_ca bus

Class
CsrRxpStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:1]	U0		Unused. Drive 0.
d0[0]	EnableEcc		Enable ECC correction

5.9.5.2 RxpCsrStat - For csr_rxp_Stat_ca bus

Class
RxpCsrStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:12]	U0		Unused. Drive 0.
d0[11]	DoubleBitErr		ECC corrector detected a double bit ECC error while reading the packet buffer or operand memory.
d0[10]	SingleBitErr		ECC corrector detected a single bit ECC error while reading the packet buffer or operand memory.
d0[9:8]	RxWhichBuf		Provide RxWhichBuf in R_SDmaPortStatus[X]
d0[7:0]	RxBufState		Provide RxBufState in R_SDmaPortStatus[X]

5.9.6 Encoding of Buses between DmaCsr and DmaCopy

5.9.6.1 CsrCopyStat - For csr_copy_Stat_ca bus

Class
CsrCopyStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:3]	U0		Unused. Drive 0.
d0[2]	EnableEcc		Enable ECC correction
d0[1:0]	FlipMemBits		XOR these bits with ALU result data before writing to packet buffer or read/write memory buffer.

5.9.6.2 CopyCsrStat - For csr_copy_Stat_ca bus

Class
CopyCsrStat

Bit	Mnemonic	Type	Definition
d1[63:0]	U1		Unused. Drive 0.
d0[63:18]	U0		Unused. Drive 0.
d0[17]	DoubleBitErr		ECC corrector detected a double bit ECC error while reading the packet buffer or read/write memory buffer.
d0[16]	SingleBitErr		ECC corrector detected a single bit ECC error while reading the packet buffer or read/write memory buffer.
d0[15:14]	CopyTxWhichBuf		provide CopyTxWhichBuf in R_SDmaCopyPortStatus
d0[13:12]	CopyRxWhichBuf		provide CopyRxWhichBuf in R_SDmaCopyPortStatus
d0[11:0]	CopyBufState		provide CopyBufState in R_SDmaCopyPortStatus

Chapter 6

Processor Segments

[$Id: processor.lyx 47578 2007-11-16 21:54:43Z wsnyder $]

6.1 Overview

The SCX1000 includes six identical processors implementing the MIPS64 Architecture including floating point. Each CPU is a MIPS 5kf with custom extensions. (MIPS may rename our re-derived CPU, but for now, we'll continue to call it 5kf.) The processor segment contains one CPU, its associated 256KB L2 cache segment, maintenance and control registers, and the processor interrupt controller.

6.2 Specifications

Each processor has the following major features, with features we've changed or configured from the base MIPS 5kf indicated in bold:

- 64-bit Data and address path
- 42-bit Virtual and 36-bit physical address space
- MIPS64 Compatible Instruction Set
  - Multiply-Accumulate and Multiply-Subtract (MADD, MADDU, MSUB, MSUBU)
  - Zero/One Detect (CLZ, CLO, DLCO, DLCZ)
  - Conditional Move Instructions (MOVZ, MOVN)
  - Prefetch Instructions (PREF, PREFX), **including L2 prefetches**
- Dual issue super-scalar architecture, capable of simultaneously executing:
  - 1 integer and 1 arithmetic floating point
  - 1 floating point arithmetic and 1 floating point store
- Floating Point
  - IEEE 754 compatible
  - Single and double precision
  - Multiply and add instruction
  - **Issue one multiply add double every clock**
  - Fast flush-to-zero mode to optimize performance
- Multiply/Divide Unit
  - Issue one 32x16 multiply every clock

  - Issue one 32x32 multiply every other clock
  - Issue one 64x64 multiply every nine clocks
  - 37 clock latency on 32/32 divide
  - 69 clock latency on 64/64 divide
  - Early-in feature returns division results sooner for smaller dividends

- Memory Management Unit
  - **48 dual-entry JTLB**
  - 4-entry instruction micro TLB
  - 4-entry data micro TLB
  - 16 KB to 16 MB page sizes. **(Note 4KB pages are not supported.)**
  - 8 bit ASID.

- Caches
  - **32 KB 4-Way Data cache**
  - **32 KB 4-Way Instruction cache**
  - **Write-back and write-allocate**
  - Non-blocking loads
  - 32-byte cache line size
  - Virtually indexed, physically tagged
  - Support for locking cache lines
  - Non-blocking prefetches
  - **ECC protected Data Cache, parity protected I Cache**

- Bus Interface Unit
  - Separate 32-bit address request bus and 64-bit data bus
  - Four 64-bit IO write buffers
  - One 32-byte eviction buffer
  - **Load Linked, Store Conditional multi-processor support**
  - **SYNC instruction support**

- **Independent intervention (probe) bus**
  - **Probing of D-Cache, Write Buffers**

- Performance Monitoring logic

6.3 User Code Visiable Bugs and Enhancements

6.3.1 Product and Chip Pass Differences

1. ICE9B returns a different product (ICE9B) when reading **R_CpuPRId** and R_CpuTapIDCODE.
2. ICE9B fixes bug1965 whereby **R_CpuErrCtl** reads swap bits 31 and 28. In ICE9A any read-modify-writes need to swap these bits before writing them back.
3. ICE9B improves **micro DTLB performance** bug 2200 with a entry size of 64KB when the corresponding TLB entry is 64KB or larger. If the TLB entry is 16KB, the old 4KB uTLB entry size is used.
4. ICE9B improves probe performance by using 64 byte probes, see bug2202.

5. ICE9B removes an unnecessary syncronizer on the cac_cpu_int wires, this reduces interrupt latency by one pclk.

6. ICE9B adds **performance counter events** for L2 misses and floating point operations, and allows all events to be visible to both counter 0 and counter 1.

7. TWC9A returns a different product (TWC9A) when reading **R_CpuPRId** and R_CpuTapIDCODE.

8. **TWC9A uses a new core, IceT. This is described in a different document.**

6.3.2 Known Bugs and Possible Enhancements (M5KF only)

1. On D-Cache ECC errors, **R_CpuCacheErr_EW** may record the incorrect way number and index**,** see bug1575. As a workaround, software should flush the entire cache on ECC errors.

2. On filling the TLB with a **4KB page**, we should pull a machine check, as 4KB pages are not supported.

3. On **writes to accelerated space**, we should pull a machine check, as they are not supported.

4. We should add a 64-bit cycle counter which is NOT writable, as the current count register is occasionally overwritten by the kernel, bug3342.

5. We should implement the RDHWR instruction so user space code can see the cycle counter and processor number.

6. We should add more VA bits, to enable the VA to be unique across the entire system.

6.4 Kernel and Performance Bugs and Enhancements

6.4.1 Product and Chip Pass Differences

1. ICE9B returns a different product (ICE9B) when reading **R_CpuPRId** and R_CpuTapIDCODE.

2. ICE9B fixes bug1965 whereby **R_CpuErrCtl** reads swap bits 31 and 28. In ICE9A any read-modify-writes need to swap these bits before writing them back.

3. ICE9B improves **micro DTLB performance** bug 2200 with a entry size of 64KB when the corresponding TLB entry is 64KB or larger. If the TLB entry is 16KB, the old 4KB uTLB entry size is used.

4. ICE9B improves probe performance by using 64 byte probes, see bug2202.

5. ICE9B removes an unnecessary syncronizer on the cac_cpu_int wires, this reduces interrupt latency by one pclk.

6. ICE9B adds **performance counter events** for L2 misses and floating point operations, and allows all events to be visible to both counter 0 and counter 1.

7. TWC9A returns a different product (TWC9A) when reading **R_CpuPRId** and R_CpuTapIDCODE.

8. **TWC9A uses a new core, IceT. This is described in a different document.**

6.4.2 Known Bugs and Possible Enhancements (M5KF only)

1. On D-Cache ECC errors, **R_CpuCacheErr_EW** may record the incorrect way number and index**,** see bug1575. As a workaround, software should flush the entire cache on ECC errors.

2. On filling the TLB with a **4KB page**, we should pull a machine check, as 4KB pages are not supported.

3. On **writes to accelerated space**, we should pull a machine check, as they are not supported.

4. We should add a 64-bit cycle counter which is NOT writable, as the current count register is occasionally overwritten by the kernel, bug3342.

5. We should implement the RDHWR instruction so user space code can see the cycle counter and processor number.

6. We should add more VA bits, to enable the VA to be unique across the entire system.

6.5 Complete Documentation

For complete information on the MIPS 5kf core, see the documentation provided by MIPS. The remainder of this chapter will discuss only the bus interface and items being changed inside the CPU.

(Tech Pubs: Remove this and insert the relevant 5KF documentation.)

6.6 BIU Description

The CPU bus interface connects the CPU with the associated L2 cache. The BIU interface is based upon the default 5kf interface, with some extensions as described below.

6.6.1 BIU Ports

Signals corresponding to original MIPS 5kf BIU signals are listed below. The capitalized middle part of the signal always corresponds to the original MIPS signal name with EB_ prepended, for example cpu_cac_reqAValid_pr corresponds to EB_AValid.

Name	In/Out	Product	Description
cac_cpu_reqARdy_pr	In		Cache ready for new address, CPU may send _reqAValid in the next
cac_cpu_reqWDRdy_pr	In		Cache ready for new write data, CPU may send write data in the ne
cpu_cac_reqAValid_pr	Out		Address bus and access type are valid this cycle.
cpu_cac_reqAddr_pr[35:3]	Out		Read/write transaction address.
cpu_cac_reqBE_pr[7:0]	Out		IO transaction byte enables.
cpu_cac_reqBurst_pr	Out		Burst transaction; reqBFirst, reqBLast and reqBLen indicate the sta
cpu_cac_reqBFirst_pr	Out		First cycle of multiple-cycle burst. May not be needed, as can be det
cpu_cac_reqBLast_pr	Out		Last cycle of multiple-cycle burst. May not be needed, as can be det
cpu_cac_reqBLen_pr[1:0]	Out		Number of cycles in burst. Not valid for non-bursts.
cpu_cac_reqInstr_pr	Out		Read is for an instruction fetch. Data will go to the I-Cache and so t
cpu_cac_reqWData_pr[63:0]	Out		Write data.
cpu_cac_reqWrite_pr	Out		Write, not a read.
cac_cpu_rtnRdVal_pr	In		Read return data is valid this cycle.
cac_cpu_rtnRBErr_pr	In		Read return is in error. (unused, tied false)
cac_cpu_rtnRData_pr[63:0]	In		Read return data.
cpu_cac_wbWWBE_pr	Out		CPU is waiting for write buffers to empty. This may be used to re-p
cac_cpu_int_p[3:0]	In		Six bit interrupt request mask. Top two bits are tied to 0.

The following signals have been added to the base design:

Name	In/Out	Product	Description
cpu_cac_reqCmd_pr[2:0]	Out	TWC9A+	Requested command. Valid when cpu_cac_reqVld_pr is asserted. See 6.26.1 on page 340.
cpu_cac_reqRId_pr	Out	TWC9A+	Requested read identifier. For reads or prefetches, this indicates which CPU read-id needs to be indicated with the eventual return and retirement.
cac_cpu_rtnPMHit_pr	In		Read return hit in L2 Cache. Valid when cac_cpu_rtnRdVal_pr asserted for cachable addresses.
cac_cpu_rtnPMState_pr[2:0]	In		Read return CacState. Valid when cac_cpu_rtnRdVal_pr asserted with cac_cpu_rtnPMHit_pr.
cac_cpu_rtnPMStop_pr[3:0]	In		Read return bus stop number. CswStopNum for memory (non IO) read data, valid when cac_cpu_rtnRdVal_pr asserted.
cac_cpu_rtnRId_pr[2:0]	In	TWC9A+	Read return identifier. When cac_cpu_rtnRdVal_pr asserts indicates which read return the data is for. This is the identifier requested with cpu_cac_reqRId_pr.
cac_cpu_rbDone_pr[7:0]	In	TWC9A+	Read buffer completion. When a bit pulses for one cycle, the corresponding cpu_cac_reqRId_pr number may now be retired and reused. If it's reused, this same number may appear on cpu_cac_reqRId_pr as soon as the cycle after next. This handshake is independent of cac_cpu_rtnRdVal_pr, as it has the flexability to hold a buffer until a TID is done, and alows multiple TIDs to retire at once.
cac_cpu_syncBusy_pr	In		Sync Holdoff. Asserted to indicate sync instructions must be held off. Must first assert two cycles after cpu_cac_reqAValid_pr & cac_cpu_reqARdy_pr are asserted, and cleared when sync instructions may complete.
cac_cpu_wbIoAck_pr	In		Pulsed to indicate a IO write buffer has been emptied on the L2 side, and a credit should be added to the buffer count.
cac_cpu_prbReq_pr	In		Probe address request this cycle.
cac_cpu_prbAddr_pr[35:3]	In		Probe address. Note wrapping request on [4:3] is only a hint, and cannot be guaranteed to be the order returned by the CPU. In fact, it is always 0. In pass2, probes are 64 bytes, and bit[5] is ignored.
cpu_cac_invAck_pr	Out		Intervention acknowledge, invDirty indicates hit state.
cpu_cac_invHit_pr[1:0]	Out		Intervention hit on cache or write buffer. L2 must not require this signal for correct protocol, it is for statistical, verification, and debugging use. In ICE9A, this is a single bit signal, in ICE9B+ it indicates the status of each 32B half of the 64B probe.
cpu_cac_invDirty_pr[1:0]	Out		Intervention hit on dirty cache or write buffer. In ICE9A, this is a single bit signal, in ICE9B+ it indicates the status of each 32B half of the 64B probe.

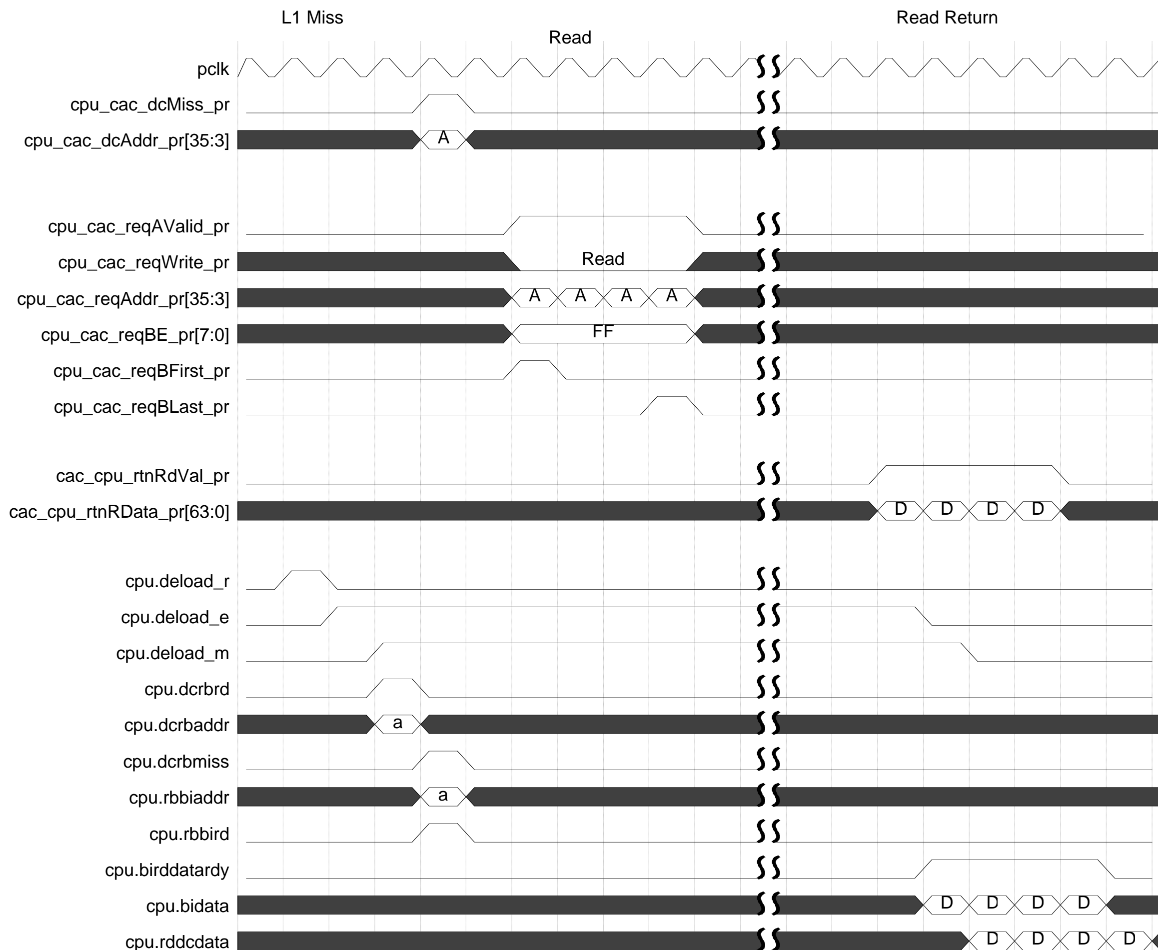


Figure 6.1: BIU Read Transaction Timing

6.6.2 D-Cache Reads

D-Cache transactions begin with a load instruction in the R stage of the pipe. The address is determined to miss in the L1 D-Cache, and the speculative miss dcMiss and dcAddr signals are asserted. The transaction is sent to the BIU. If there was dirty L1 data to be evicted, it is extracted and added to the write buffer, and becomes a write transaction described below.

The BIU issues the read request to the L2 by asserting reqAValid_pr with a burst length of 4 (there are four 64-bit chunks in the 32B cache line.) When the L2 completes the request, the L2 places the four data bursts on rtnRData_pr, and asserts rtnRdVal_pr with the read identifier on rtnRId_pr. The return order of data must match that requested. When the TID is completed, the L2 asserts rbAck_pr with the read identifier on rbRId_pr.

If the processor attempts a DCache read to a block in the SHARED state, the L2 lookup will result in a MISS. This will cause the SHARED block matching the target address to be "victimized" (that is, replaced in the L2) and a RDEX to be issued to the CSW to fill the block from main memory.

6.6.3 I-Cache Reads

Instruction cache reads look the same to the L2 cache as data stream reads. The CPU indicates the read is for I-Stream by asserting reqInstr_pr along with the address. The L2 may use this to fill the L2 cache in shared state. Since interventions do not probe the I-Cache, instruction lines may be in multiple CPU I-Caches simultaneously.

Istream accesses to L2 cache blocks in EXCL, DIRTY, or UPDATED states will result in an L2 cache hit.

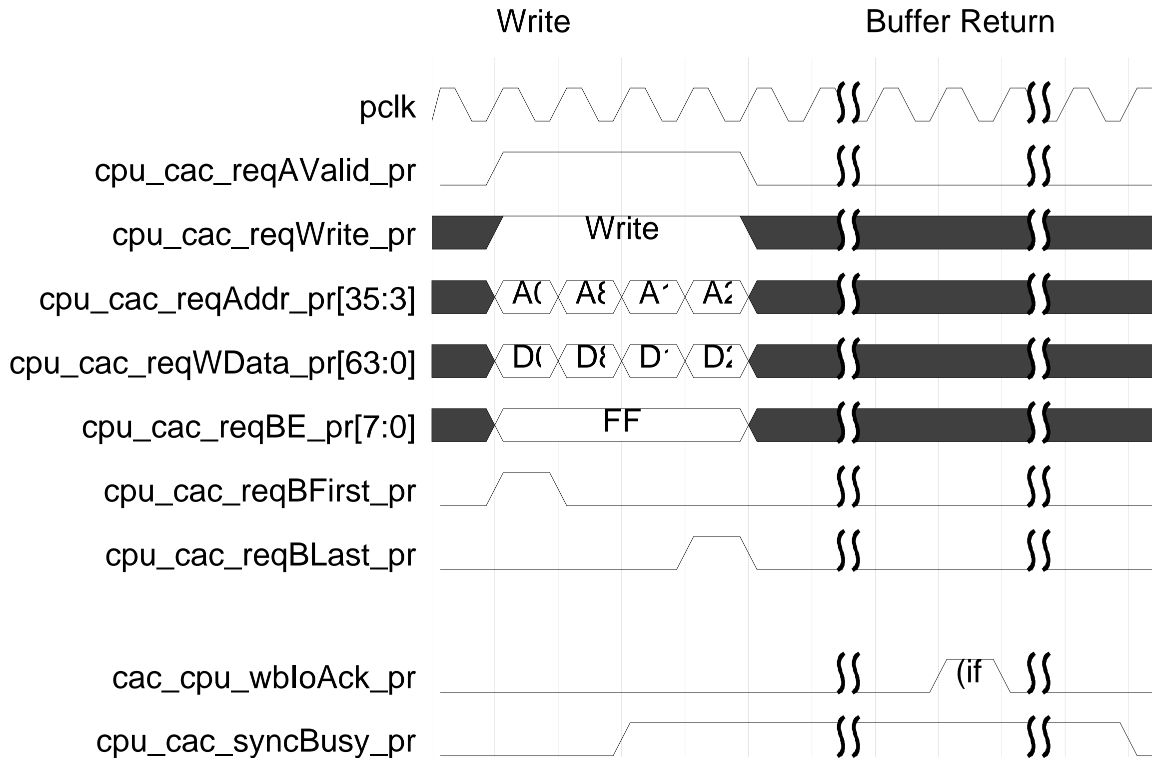


Figure 6.2: BIU Write Transaction Timing

6.6.4 Istream Initial Reads

The L2 cache supports I-Stream accesses while the L1 cache was disabled. This allows booting of the processor, and cache trap handlers which enter non-cachable mode.

6.6.5 Evictions

L1 evictions are handled by the standard MIPS interface. When a cache fill is required, the LRU line from the cache is read out and stored into the BIU write buffer. After the BIU places the read request on the bus, the eviction is requested, and the write data transferred. The L2 must assert syncBusy_pr one cycle after the write is received, and keep it asserted until the write is coherent, see 6.6.9.

To prevent deadlock, the L2 cache must accept any number of evictions while a probe is outstanding. Evictions should thus always be able to be written back to the L2, and should never require Coh action (and thus potential deadlock.)

6.6.6 IO Writes

IO Writes are handled by the standard MIPS interface. IO Writes are distinguished by address bit [35] being set. The BIU places the write on the bus. The L2 must assert syncBusy_pr one cycle after the write is received, and keep it asserted until the write is coherent, see 6.6.9. The ICE9 chip does **NOT** support “accelerated uncached write bursts” from the MIPS core. The L2/CSW supports only one active IO write at a time, so IO writes are enqueued in the interface between the MIPS core and the CSW. (See Section 6.18.)

6.6.6.1 IO Write Buffer Counter

To prevent overrunning the write buffer in the L2 cache, the BIU keeps track of the number of L2 IO write buffer entries that may be in use. The count starts at 5 entries, the size of the CPU and L2 write buffer. As IO write buffer entries are allocated, the count is decremented, where a IO write is defined as a write with address bit [35] set. When a IO write reaches the L2 coherency point, the L2 asserts wbAck_pr, which increments the count.

If the write buffer count minus the number of load/stores in flight is less than 2, on the next load/store the instruction pipeline stalls until a buffer is freed. (The extra buffer is due to pipeline delays in decrementing versus checking the count, covering the case of when there are back-to-back stores.)

6.6.7 Cache Instructions

The CPU implements the MIPS CACHE Instruction. The L1-D "hit writeback" cache instruction has been changed to instead perform "hit writeback and invalidate." This prevents the L2 from seeing an eviction from the cache instruction and believing it is the probe return. (Thus, we can enforce the rule that after eviction, a line is always invalid.)

6.6.8 Prefetch Instruction

The CPU implements the MIPS PREF Instruction.

ICE9 used the original core, which implements load and store hints identically, and the writeback invalidate hint. Prefetches issued when the cache pipeline was busy were silently dropped.

TWC9 prefetches are not dropped when the cache pipeline is busy, however they are still dropped on a TLB miss; they never take exceptions. TWC9 also adds L2 prefetches, see 6.24.4 on page 319.

TWC9 retains the rule that there may be only one miss at once. However, there may be as many as 4 misses and L2 prefetches outstanding. In addition, a second miss-under-miss will be automatically converted into an L2 prefetch. This allows software to get most of the latency benefit of two misses outstanding even if prefetches have not been inserted into the code.

For L2 prefetches, TWC9 issues a PREF command on cpu_cac_reqCmd_pr. Data is never returned. When the prefetch completes, the L2 asserts cac_cpu_rbAck_pr, with cac_cpu_rbRId_pr indicating which prefetch has completed.

Note prefetches are not supported to DMSEG when in Debug mode, the behavior is unpredictable. It's assumed there won't be any prefetches in the debug handler.

6.6.9 Sync Instruction

The SYNC instruction requires all loads and stores that occurred before the SYNC to be completed before any loads or stores following the SYNC. In our multiprocessor system, this requires all loads to be completed and have results in the register file, that all cacheable stores have invalidated other CPUs caches, and that all non-cacheable I/O stores have reached the point at which they are ordered with respect to all other CPUs.

Load/Sync ordering is be insured by stalling any SYNC until all loads have reached the register file. The original CPU has code for this, but it should be verified.

Cached Store/Sync ordering is insured by the L2 Cache asserting cac_cpu_syncBusy_pr until all stores have completed, including invalidating the caches of other CPUs.

IO Store/Sync ordering is also be insured by stalling the SYNC until cac_cpu_syncBusy_pr. syncBusy must remain asserted until the IO store has reached the IO write coherence point.

6.6.10 Load Linked and Store Conditional

The Load Linked (LL, sometimes also called Load Locked) and Store Conditional (SC) instructions are used to implement critical sections. A LL instruction loads a memory location, remembers the address loaded and sets the lock bit. The following SC returns the lock bit to the register file, and if the lock bit was set, performs a store. Any store or DMA write (not just a SC completing) to the same address causes the lock bit to clear.

To implement this scheme, we take a simple approach; we prevent any other processor from gaining access to the locked line for a certain holdoff time.

- On executing a LL, we set the lock bit, and save the locked address. We start a timer, the Locked timer, which counts up to 8 then resets. (Programmable from 8 to 1K in powers of two with R_CpuConfig_LLTime.)
- On executing a SC, we test the lock bit, and reset the locked timer.
- On executing a ERET, we clear the lock bit.

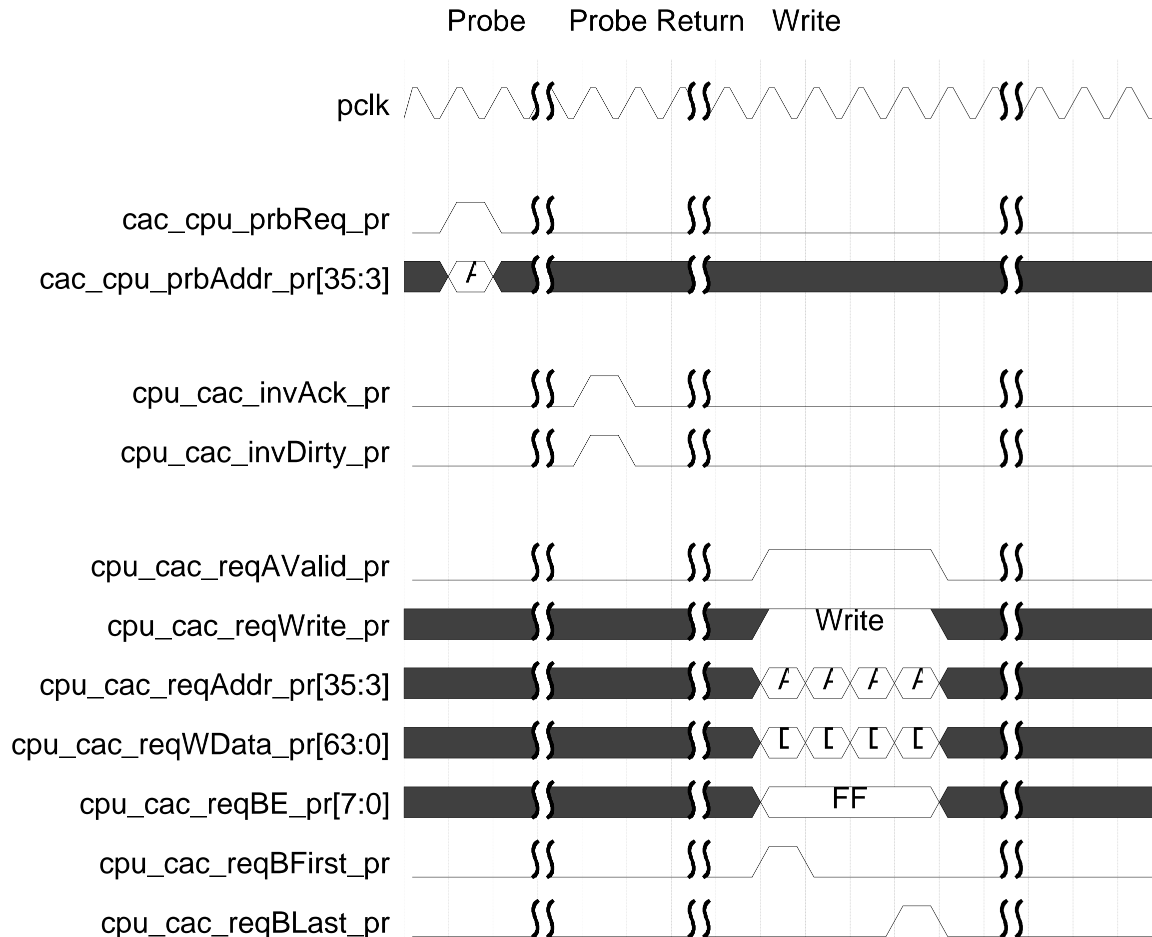


Figure 6.3: BIU Probe Timing

- While the locked timer is counting, all probes will be held off, and the CPU is free to (hopefully) complete the lock sequence. Note 8 cycles is enough to complete all Linux locks, and other locks we know about. Should the lock complete, or the SC never execute, all is fine, otherwise:
- If a probe occurs outside the locked timer interval, and the probe address matches the lock address, the lock bit is cleared.
- To prevent code that does LL inside a tight loop from livelocking out other CPU's probes forever, after the locked timer has been used for N cycles, the lock timer will not work for another N cycles. A SC is still likely to be succeed during this time; however it is not guaranteed to succeed as it otherwise would.

Note at all times lock semantics are preserved; there is no case where write data could interfere with the critical section.

Should software have large lock sequences over 8 instructions, there may be performance problems. To mitigate this, we make the interval programmable, and have an SCB event to track clearing of the lock due to probes.

6.7 Interventions

The CPU has an intervention bus to maintain coherency between the cores. The bus runs at processor clock frequency, and consists of an address, command, and acknowledgment back to the L2.

The intervention bus is presented with an address from the L2 cache. First, if any load/stores are in the pipeline, the pipeline is stalled.

This intervention address is looked up in the L1 tag store array. A clean hit will invalidate the line in the L1 D-Cache. A dirty hit will stall the load/store pipeline, and grab the D-Cache for four cycles. The data is read from

the D-Cache in 0-3 order and placed into the CPU's eviction buffer. This requires the evicion buffer be free; if not, the extraction stalls until space becomes available.

The intervention address is also compared against the CPU load/store buffer, this insures data is returned for hits on stores waiting for the L1 cache. A match will return dirty hit, and the L2 is responsible for retrieving the data from the stream of write data.

The intervention address does not need to be compared against read requests. A match against an unissued load can be ignored, as when it finally issues in the L2, the data will have been returned. A probe will not be issued against a issued load, as this is guarded by the L2 line-collision CAM.

6.7.1 Intervention Deadlock Avoidance

The intervention scheme requires that the load/store and eviction buffers makes forward progress; however the buffers may contain write transactions that have not yet reached the L2 cache and thus are before the coherency point. To prevent this resource loop from resulting in a deadlock, the L2 must insure that CPU reads and writes can always be drained. When the L2 is accepting transactions, (that is, when it is asserting ARdy) it will accept and process L1 writebacks and all other writes in order and without queuing. If necessary in handling probes the L2 interface will enqueue cache read operations for processing after the completion of writeback or probe operations. The space required for the "pending read queue" is relatively small, as the processor is limited to just two outstanding READ operations at a time. (See 6.15.3.)

6.7.2 Example Intervention Cases

Case	Actions
1. Not in D-Cache 2. Intervention	The CPU acknowledges the intervention as a miss.
1. Clean in D-Cache 2. Intervention	The CPU acknowledges the intervention as clean and invalidates the D-Cache.
1. Dirty in D-Cache 2. Intervention	The CPU acknowledges the intervention as dirty, reads the data from the cache and places into the write buffer. The write is made to the L2.
1. Miss in progress, not issued by L2 2. Intervention	The CPU acknowledges the intervention as a miss. This is correct, as the CPU miss is ordered after the intervention.
1. Miss in progress, issued by L2, data not to CPU yet 2. Intervention	Illegal. As the L2 has not returned the data, the L2 is required to stall issuing the intervention until it does so.
1. Miss in progress, issued by L2, data sent to CPU 2. Intervention	The CPU stalls the intervention on read data buffer hit until the miss updates the L2, and then the intervention becomes a L1 hit case.
1. In write buffer 2. Intervention	The CPU acknowledges the intervention as a dirty hit. The write will propagate to the L2 as writes normally do.
1. In D-Cache 2. Load or store in M or W-stage 3. Intervention	The CPU stalls the intervention until the load or store completes; additional loads or stores will stall if to the same D-Cache index. (The physical address is not known in time, and the index is identical between the VA & PA.)

6.8 WAIT

The CPU includes the WAIT instruction which places the CPU into power down mode until an enabled interrupt occurs (generally, this is a timer interrupt that was configured just before entering sleep.) During sleep, the BIU will awaken to accept and return interventions, identical to normal awakened mode.

6.9 Interrupts

The CPU provides 6 level sensitive interrupts. (It also has a non-maskable interrupt or NMI that is unused.) These first four of the six are activated by writes to the interrupt control register, the arrival of a slow interrupt, or via a CSW INT transaction. (See Section 7.10.5 and Sections 7.18.6 through 7.18.9.) The top two levels are reserved for causes internal to the processor.

Interrupt	Pin	Description
7	int[5]	Cac ICR10/11 and CPU core performance counter interrupts.
6	int[4]	Cac ICR7/8 and R_CpuCompare timer interrupts.
5	int[3]	Cac ICR6/7 and slow interrupts.
4	int[2]	Cac ICR4/5, generally DMA.
3	int[1]	Cac ICR2/3, generally PCI-E.
2	int[0]	Cac ICR0/1, generally interprocessor interrupts.
1	N/A	Software interrupt from same core.
0	N/A	Software interrupt from same core.

6.10 EJTag

The MIPS EJTAG port is connected to the SysChain JTAG bus so that the cores may be debugged. In addition a syschain register allows a debug trap on one CPU to cause debug traps to be taken on all CPUs.

6.11 D Cache ECC

The D-Cache has been changed to use byte ECC instead of byte parity. This was done without changing the pipeline or any instruction timings.

6.12 Scheduling Hazards

The CPU has the same instruction hazards as documented in the M5KF Software Users Manual, Section 12.2, with the following exception.

The original 5KF required a CACHE instruction not be followed by a memory operation for 2 instructions. This restriction is removed, any instruction may follow a CACHE instruction, including a load/store to the same cache line.

6.13 Dual Issue

The CPU has the same dual issue rules as the 5kf. As its documentation is a bit obtuse, here is a restating of the rules.

Dual issue if all of the following are true:

- Not in delay slot.
- Single-issue bit is off.
- The instruction will not trap. (IE to dual issue a COP1 instruction, COP1 must be enabled.)
- One of the pair of instructions is: abs.*, add.*, c.*, ceil.*, cvt.*, div.*, floor.*, madd.*, mov.*, movcf, msub.*, mul.*, neg.*, nmadd.*, nmsub.*, recip.*, round.*, rsqrt.*, sqrt.*, sub.*, trunc.*, MMDX with instr[5:0]!=6'b0110x1, or COP2 instruction with instr[25]=1'b1. (Note this excludes ldxc1, luxc1, lwxc1, movz, movn, prefx, sdxc1, suxc1, swxc1.)
- The other of the pair of instructions is: add, addi, addiu, addu, and, andiori, break, cache, dadd, daddi, daddiu, daddu, ddiv, ddivu, div, divu, dmfc1, dmtc1, dmult, dmultu, dsll, dsll32, dsllv, dsra, dsra32, dsrl, dsrl32, dsrlv, dsub, dsubu, lb, lbyu, ld, ldc, ldc2, ldl, ldr, ldxc1, lh, lhu, ll, lld, ltl, lui, luxc1, lw, lwc1, lwc2, lwl, lwr, lwu, lwxc1, mfc1, mfhi, mflo, movn, movz, mtc1, mthi, mtlo, mult, multu, or, pref, prefx, sb, sc, scd, sd, sdc1, sdc2, sdl, sdr, sdrav, sdxc1, sh, sll(excluding_nop), sllv, slt, sltiu, sltu, sra, srav, srl, srlv, stti, sub,

subu, suxc1, sw, swc1, swc2, swl, swr, swxc1, sync, syscall, teq, tge, tgeu, tltu, tne, xnor, xor, xori, or COP2 instruction with instr[25:22]==4'b00x0. (Note this excludes cfc1, ctc1, deret, eret, jr, jalr, mfc0, movci, mtc0, ssnop.)

6.14 Floating Point Pipeline Enhancements

The floating point pipe was modified to increase the issue rate of double-precision multiply and fused-multiply-add instructions. These include mul.d, madd.d, msub.d, nmadd.d, & nmsub.d. The effect is to change the m5kf latency (5 cycles) and "issue rate" (2 cycles) for these instructions to 4 cycles & 1 cycle, matching the latency and "issue rate" of the corresponding single-precision version of the same instructions. As a side effect of the change, recip.d and rsqrt.d also come out with improved performance.

In the original m5kf, the resources devoted to the multiplier array were reduced (optimized) by implementing half the hardware needed for a full double-precision multiplier and using the hardware on 2 consecutive cycles to complete a double-precision multiply. (Single-precision multiply operations don't need the additional cycle, so they complete the multiply part of the operation in 1 cycle.) As a result, a multiply instr. following a d.p. multiply had to wait a cycle before issuing, since the hardware would still be in use for the 2nd cycle of the preceeding multiply instruction. By building the full hardware need for a d.p. multiplier, the issue rate was doubled and the latency reduced, for something like a 10-15% improvement in delivered performance.

The aproach we've taken in implementing the ICE9 changes is to collapse 28 booth partial-products, plus 2 injected constants into the sum-and-carry redundant-form representation of the multiply result in a single cycle. This requires 4 levels of CSA, one more than in 1 cycle of the m5kf multiplier. The additional CSA inserted into the cycle adds to the critical path in the multiplier array, but there was sufficient margin to make the insertion without impact to the chip clock frequency. The changes are illustrated in the following 2 figures. The first shows the organization of the m5kf multipler array. The 2nd shows the organization of the ICE9 multiplier array.

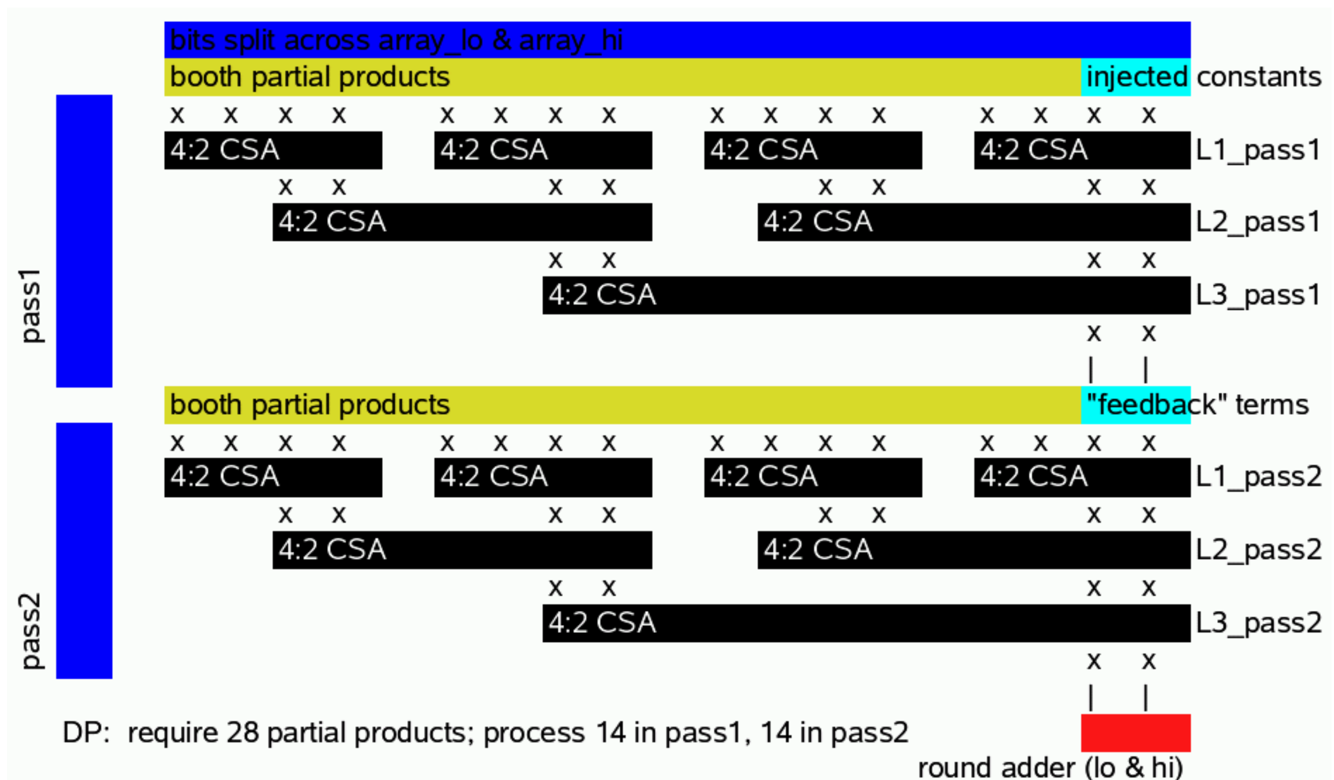


Figure 6.4: M5kf Multiplier

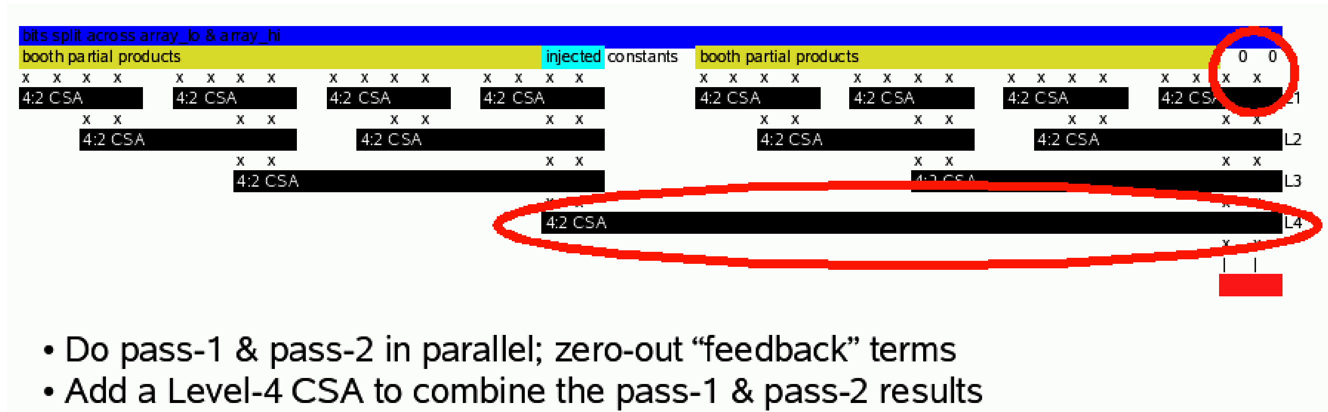


Figure 6.5: ICE9 Multiplier

6.14.1 Floating Point Repeat Rate and Latency

Bolded values indicate change from M5KF.

Opcode	Latency (cycles)	Repeat Rate (cycles)
ABS.*, NEG.*, ADD.*, SUB.*, MUL.*, MADD.*, MSUB.*, NMADD.*, NMSUB.*	**4**	**1**
RECIP.S	**15**	**10**
RECIP.D	**23**	**18**
RSQRT.S	**19**	**14**
RSQRT.D	**31**	**26**
DIV.S, SQRT.S	17	14
DIV.D, SQRT.D	32	29
C.cons.* to MOVD.* and MOVT.*/ MOVT, MOVN, BC1	1/2	1
CVT.D.S, CVT.[S,D].[W,L]	4	1
CVT.S.D	6	1
CVT.[W,L].[S.D], CEIL.*, FLOOR.*, ROUND.*, TRUNC.*	5	1
MOV.*, MOVD.*, MOVN.*, MOVT.*, MOVZ.*	4	1
LWC1, LDC1, LDXC1, LUXC1, LWXC1	3	1
MTC1, DMTC1, MFC1, DMFC1	2	1

6.15 The L2 Cache Segment and Pipelines

Each processor in the ICE9 chip is directly connected to a 256KB L2 cache segment. All six cache segments are kept coherent via the Cache Switch interface (CSW) described in Chapter 7. Most of the L2 cache (CAC) runs at the central CCLK rate, only the interface to the processor contains elements clocked on the processor clock (PCLK).

The CAC talks to the processor through the SLC unit. The SLC is responsible for retiming processor requests from PCLK to CCLK and retiming responses in the opposite direction. It processes all write requests as they are issued by the processor, and may enqueue read requests if necessary. All read requests are processed in order: reads don't pass reads. Similarly, all writes are processed in order. However, to correctly handle the case of a Dstream L1 miss that requires a victimization of an L1 or L2 block followed by an Istream L1 miss, we allow writes to pass reads.

The CAC also connects to the CSW. Probes are handled in order of arrival, but may be enqueued for an arbitrary number of cycles.

6.15.1 The Tag Lookup

The L2 segment is optimized to handle DCache misses in the absolute minimum number of cycles. Figure 6.6is a sketch of the pipeline from tag lookup to CSW command generation. From the delivery of the dcache miss address

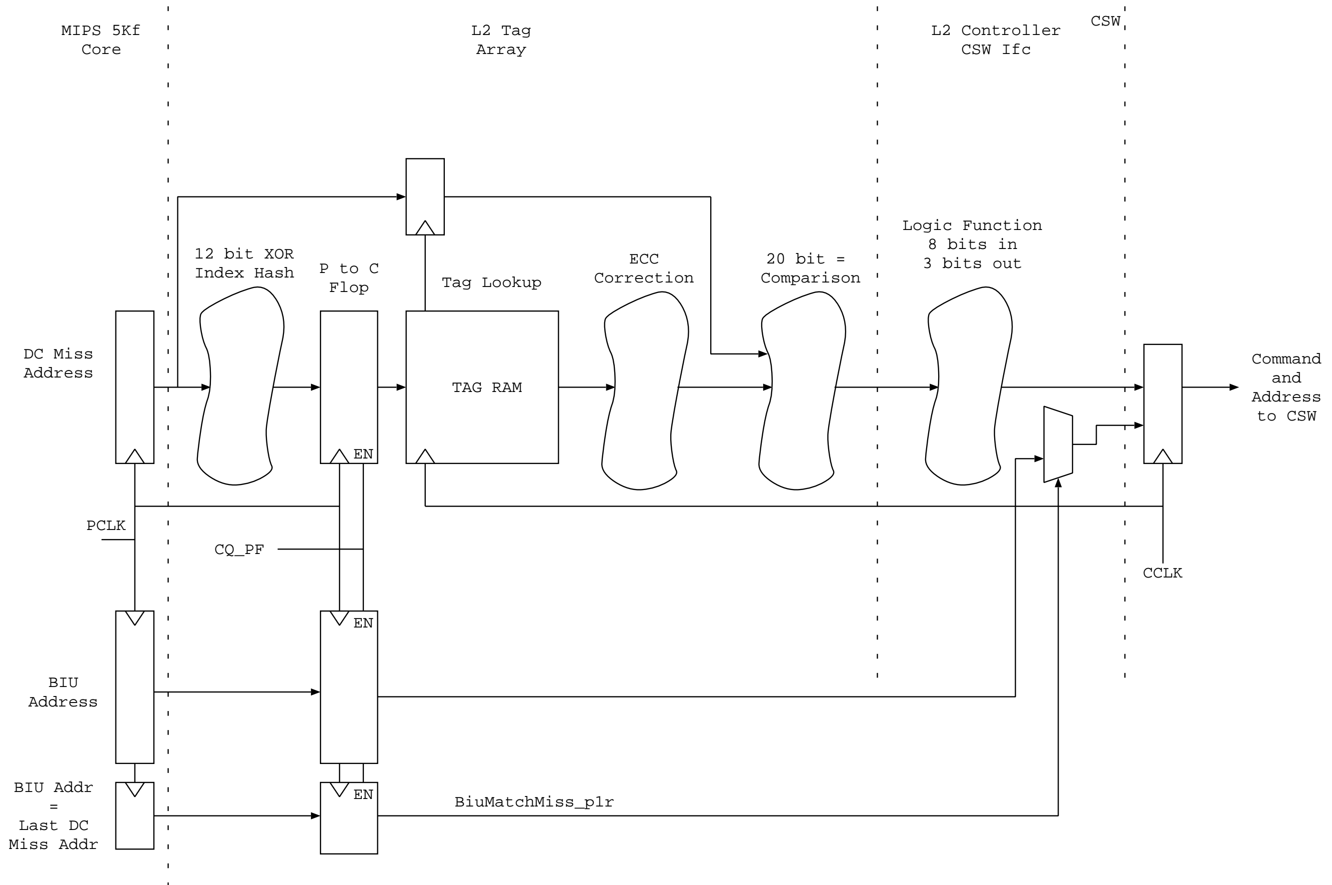


Figure 6.6: L2 Tag Lookup Pipeline

at the edge of the MIPS core (a stage that we'll label P0) to the command out to the CSW the path takes two pipeline stages of 2nS each, plus a possible realignment penalty of 2nS (to align the PCLK request with the CCLK domain) plus 4nS for the tag lookup, and a 4nS stage for driving the command to the CSW.

The last stage of logic in the lookup pipeline determines whether the PS needs to do a memory read, what kind of read command the PS should issue, and where it should go. Algorithm 6.1 describes the policy for chosing which command to issue and which way to victimize. Note that we don't wait to find out if the victim block is really dirty (requires a writeback), but instead assume that all blocks in the EXCLUSIVE, MODIFIED, or UPDATED state require a writeback. While we're launching the CSW request we'll start an L1 cache probe operation to acquire the dirty data (if any). If the displaced block is dirty, we'll drive the data onto the CSW when it is ready. If we find that the block was not dirty in the L1 cache AND it was clean in the L2, we'll send a WBCANCEL command to the appropriate coherence widget.

Figure 6.7 shows the pipeline and general organization of the L2 Tag and State arrays. All components in this section run off the central clock. Note the four way mux at the top of the pipeline. Addresses enter from either the processor BIU, or the CSW fill and probe path. The address path from the BIU is required to support flush and writeback operations and for I-stream fetches.

The Tag arrays are ECC protected. Each array contains 2K words of 26 bits each. The actual tag is 18 bits wide (address bits 34 through 17). The state information requires 3 bits. For the 20 data bits, we'll require 6 bits of SECDED ECC. The two banks are independently corrected to allow for independent updates. If the two tags are merged, the total storage requirement would be 2K words by 47 bits. Corrected words are not written back to the array. In the event of an ECC error, the L2 controller will signal an ECC error interrupt to the processor and the processor will initiate a flush of the L2 cache. Double bit errors will signal a machine check.

A block in the L2 is in one of five states:

INVALID: No data is stored in the associated block. All tag comparisons against this block will fail to match.

EXCLUSIVE: This block was filled in response to a DCache miss. The data in the block is identical to the copy

Algorithm 6.1 L2 Lookup Pipeline – CSW Command Generation

```
if (miss address is I/O space) {
  issue RDIO or WTIO as appropriate, to the ■correct■ bus stop.
  // see Section 6.18
} else if ((Way0Miss AND Way1Miss) OR
           (DFETCH AND (Way0Hit AND (Way0State == SHARE)) OR
                        (Way1Hit AND (Way1State == SHARE))){
  csw address = miss address
  select victim as per table 6.1 OR by the ■DFETCH to SHARE■ rule below.
  cmd way = victim way
  if address<6> csw destination = COHO
  else csw destination = COHE
  if (access is IStream) {
    if (victim state is SH or INV) csw command = RDSH // istream read with no write-
back
    else csw_command = RDSV // read with a possible victim writeback
  }
  else {
    if (victim state is SH or INV) csw command = RDEX // dstream read with no write-
back
    else csw_command = RDV // dstream read with possible victim writeback
  }
  bid for the appropriate CSW chain.
}


DFETCH to SHARE victimization rule:
if (DFETCH AND (Way0Hit AND (Way0State == SHARE))) victim = Way0
else if (DFETCH AND (Way1Hit AND (Way1State == SHARE))) victim = Way1
else find victim in 6.1.
```

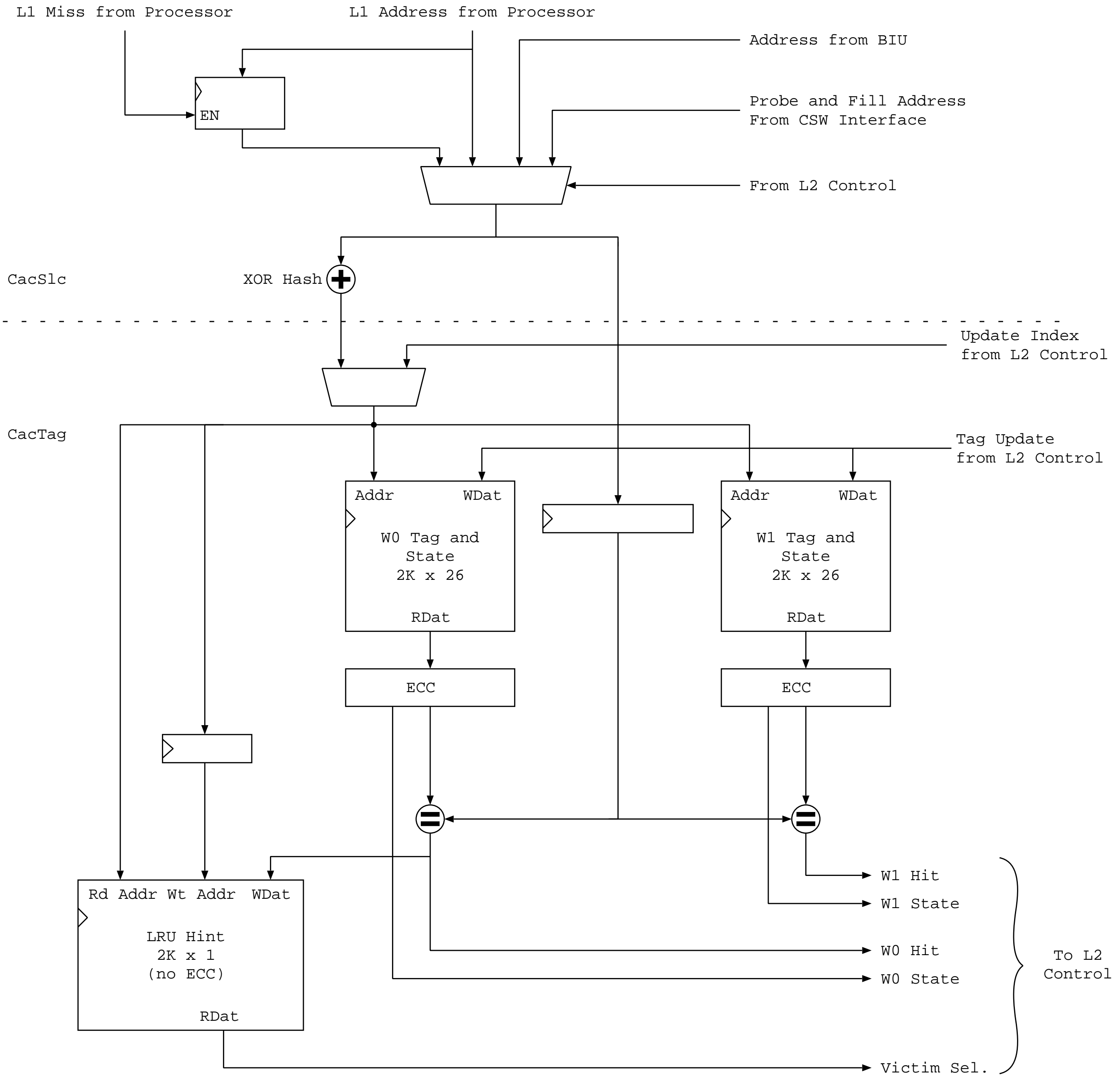


Figure 6.7: L2 Tag and State Arrays – The Address Pipeline (All in CCLK domain)

		Way 1 State				
		INV	SH	EX	MOD	UPD
Way 0 State	INV	W0	W0	W0	W0	W0
	SH	W1	LRU	LRU	LRU	LRU
	EX	W1	LRU	LRU	LRU	LRU
	MOD	W1	LRU	LRU	LRU	LRU
	UPD	W1	LRU	LRU	LRU	LRU

Table 6.1: Victimization Rules

of the data in main memory. The L1 cache may have a copy of the data that is newer still.

MODIFIED: This block was filled in response to a DCache miss. The data in the block is newer than the copy in main memory. The L1 cache may have a copy of the data that is newer still.

UPDATED: This block was filled in response to a DCache miss. Since the block was filled, the L1 cache has written data through to this block. The L1 cache may have a copy of the data that is newer still.

SHARED: This block was filled in response to an ICache miss. It is identical to the copy of data in main memory.

Note that the LRU array is a bit vector. Bit X in the vector is set if the last access to set X in the tag array hit on way zero. The L2 control unit uses this hint to chose the victim block when replacement is required. Replacement ordering rules chose the victim block on a priority basis as shown in Table 6.1. LRU is used for the replacement choice for all cases where both blocks are in a state other than INValid.

6.15.2 The L2 Miss Data Pipeline

Figure 6.6 and what we've discussed so far gets us to the command port of the CSW. The memory request will then wind its way to the memory controller and either cause a memory fetch or get forwarded to a processor that owns a copy of the block. When the data returns it will pass through the L2 update and L1 fill pipeline shown in Figure 6.8. There isn't a whole lot to do in this path. We need to grab the data from the CSW, check and correct for any single bit errors, and then forward the data into the BIU port on the processor.

The LfBuf in Figure 6.8 holds the fetched 64 byte block. The first 32 bytes are forwarded to the SLC unit and retimed to be sent into the processor. All 64 bytes are held until they are written into the L2 data array.

Figure 6.8 ommits a whole lot of detail. The L2 data array does not show details of the mux control, the L1 to L2 update path, or the address multiplexing for the L2 data arrays. None of these is all that important to the speed of L2 miss handling.

6.15.3 L1 Updates Writebacks and Misses

So far, we've described the path of L2 miss transactions. In all likelihood, at least two out of three accesses to the L2 cache will hit. Further, the L1 will occasionally displace dirty blocks into the L2. (Note that the processor will never write an L1 data block to the L2 unless it had first read the block into the L1. This means that L1 writes to the L2 will *always* hit in the L2 (since the L1 is a subset of the L2).

On an L1 read miss, the L1 may need to displace a block from the 32KB 4-way L1 DCache. Further, the read miss may require that we displace a block from the L2 as well. This means that the original L1 read miss (a single 32 byte read transaction) may cause a 32 byte writeback (the L1 victimization), *two* L2 to L1 probe operations (to find out if either of the 32 byte halves of the displaced L2 block are cached in the L1) and between zero and two 32 byte writeback operations (L1 copies of the displaced L2 block.) Confusing? Let's try a few scenarios.

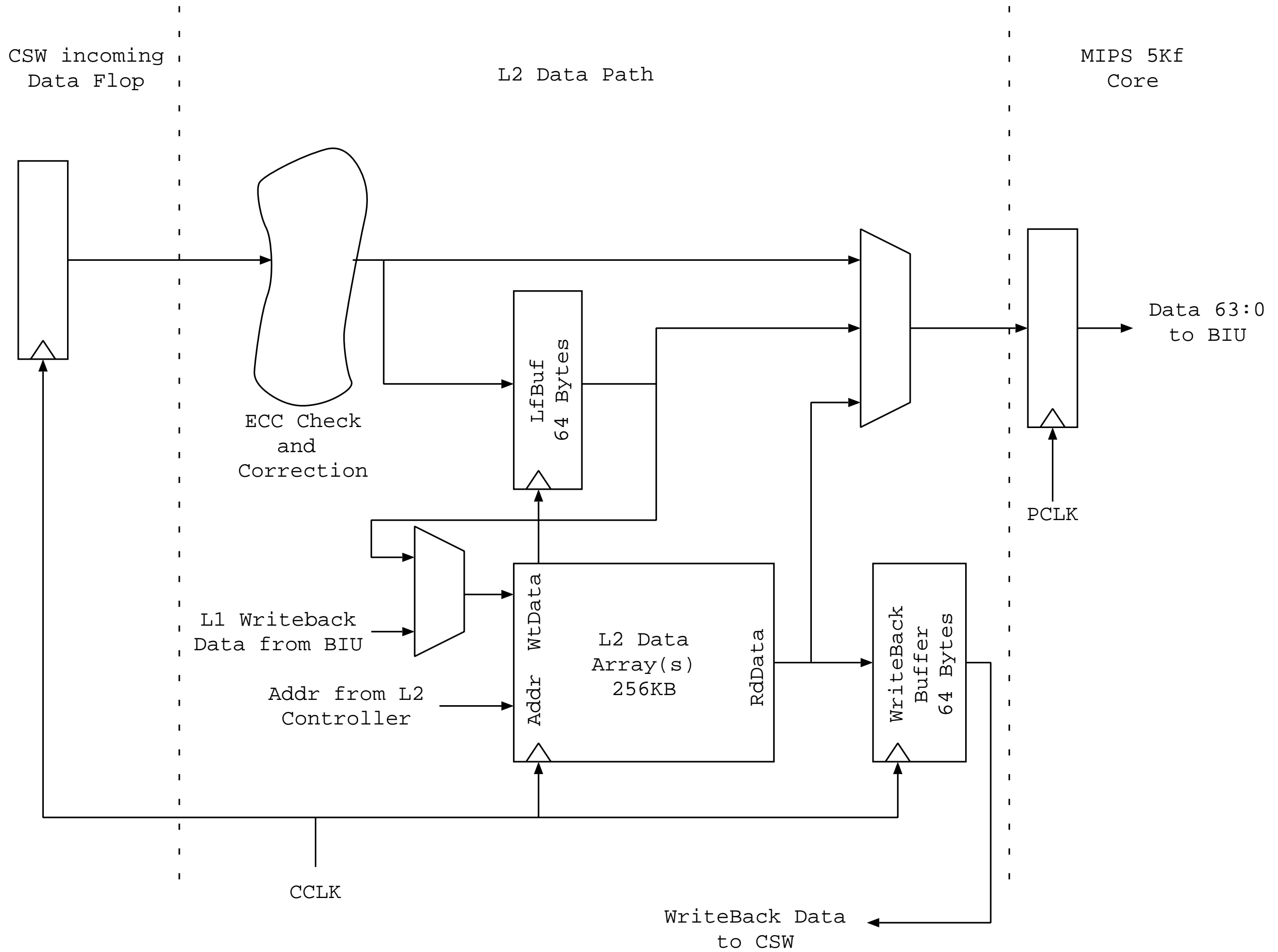


Figure 6.8: The L2 Update and L1 Fill Pipeline

Time	Operation
0	Processor issues Dstream Read of address X at BIU port
2	SLC retimes BIU request, sends address to TAG and DAT arrays
6	Tag array looks up address X. Data array begins data lookup
10	Tag is a HIT on way 0. Data array muxes set 0 data back to SLC
14	SLC retimes to PCLK domain, returns first 64 bit data word to BIU port
16	SLC returns second word to BIU port
18	Third word.
20	Fourth word.

Table 6.2: Simple L1 Read Miss – L2 Hit

Time	Operation
0	Processor issues Dstream Read of address X at BIU port
2	SLC retimes BIU request, sends address to TAG and DAT arrays. Sends data to DAT array.
6	Tag array looks up address X.
10	Tag is a HIT on way W. Data array writes first and second data words into way W.
14	Data array writes third and fourth data words into way W.

Table 6.3: Simple L1 Writeback (All L1 writes hit in L2)

Table 6.2 shows the trajectory of an L1 Dstream miss that hits in the L2. Istream misses are processed identically. Note that because of alignment issues between the PCLK and CCLK domain, the actual time line may be shifted 2 nS later (that is, SLC retiming may happen at time = 4nS) for half of all accesses.

Figure 6.6 shows the flow for an L1 read miss that requires eviction of an L1 cache block *and* an L2 cache block. Note that in this case the L1 block could map to the L2 block. (This may be impossible, given that we're using a different hash function in the L1 and L2 caches, but I'm not ready to bet on that yet.) The DAT unit ensures that any writes arriving from the processor will be checked against the L2 victim address. Writes to the L2 victim block will be routed to the WriteBack buffer (and thence to the CSW when the victim data is finally evicted).

The time between issuing a probe request into the processor's BIU and the arrival of the response can't be determined *a priori*, so the table shows the first of two probes completing at time P1. The writeback of the L1 victim block may occur at any time between the arrival of the read-miss request and *the end of time*, but the overall operation will not be complete until both probe requests have completed AND any blocks that the L1 probe identified as dirty have been loaded into the WriteBack buffer and sent out to the CSW.

6.15.4 CSW Probe Operations

From time to time the coherence engines on the CSW will forward probe requests to the PS. Each request is first processed by the L2 controller to check for collisions against operations that are currently in flight. Commands are processed in order, but not necessarily immidiately. The L2 controller queues up to 26 operations in the incoming command queue. Probes are only processed when there are *no* L2 operations in flight – this is to prevent the huge tree of possible interactions between probes and L1/L2 references.

The L2 controller then sends each probe request to the L2 tag array. In this case, the input to the tag array address mux is preempted. (This is why we capture the last DC miss address – we'll launch the DC tag query when the probe is complete.) If the L2 tag compare indicates a MISS, the controller will send a PROBENOHIT as appropriate. If the L2 tag compare hits, then we'll send a L1 probe request to the core. On completion of the core intervention, the controller will update the L2 data block with L1 writeback data and send the L2 data out to the CSW as necessary. (This latter operation is identical to a victim eviction with an L1 merge and uses the same buffers and machinery.)

Probe operations are described in detail in Section 6.22.

Time	Operation
0	Processor issues Dstream Read of address X at BIU port
2	SLC retimes BIU request, sends address to TAG and DAT arrays
6	Tag array looks up address X. Data array begins data lookup
10	Tag is a miss on both ways. Way W is selected as victim. Data array muxes data from way W (all 8 words) into the Writeback Buffer.
14	Drive RDEX (Dstream) or RDS (Istream) onto CSW as appropriate.
T	CSW returns first 16 bytes (DAT[0], DAT[1]) of data to Data array Fill Buffer
T+4	CSW returns DAT[2], DAT[3] to Fill Buffer. SLC retimes DAT[0], DAT[1] to processor BIU. DAT[0], DAT[1] written to data array. (This may be delayed if the L2 data array is busy.) Update TAG array with current MOD STATE.
T+8	SLC retimes DAT[2], DAT[3] to processor BIU. DAT[2], DAT[3] written to data array.
T+12	DAT[4], DAT[5] written to data array.
T+16	DAT[6], DAT[7] written to data array.

Table 6.4: L1 Read Miss, L2 Read Miss, Victim block is in INVALID or SHARE state

Time	Operation
0	Processor issues Dstream Read of address X at BIU port
2	SLC retimes BIU request, sends address to TAG and DAT arrays
6	Tag array looks up address X. Data array begins data lookup
10	Tag is a miss on both ways. Way W is selected as victim. Victim block address is V. Data array muxes data from way W (all 8 words) into the Writeback Buffer.
14	Drive RDV (Dstream) or RDSV (Istream) onto CSW as appropriate. Send probe for block V to processor BIU
P1	Probe completes in processor – invalidate the block, returns DIRTY if block must be written back. Writeback operations from BIU to L2 data array begin after probe response. SLC retimes writeback data, inserts data into WriteBack buffer. (Overwrites L2 data.) Send probe for block V+32 to processor BIU
P2	Probe completes in processor. If neither block is DIRTY, send WBCANCEL to CSW.
P2+4	Dump WriteBack buffer to CSW to complete RDV or RDSV writeback portion.
T	CSW returns first 16 bytes (DAT[0], DAT[1]) of data to Data array Fill Buffer
T+4	CSW returns DAT[2], DAT[3] to Fill Buffer. SLC retimes DAT[0], DAT[1] to processor BIU. DAT[0], DAT[1] written to data array. (This may be delayed if the L2 data array is busy.) Update TAG array with current MOD STATE.
T+8	SLC retimes DAT[2], DAT[3] to processor BIU. DAT[2], DAT[3] written to data array.
T+12	DAT[4], DAT[5] written to data array.
T+16	DAT[6], DAT[7] written to data array.

Table 6.5: L1 Read Miss, L2 Read Miss, Victim block is EXCL, DIRTY, or UPDATED

Time	Operation
0	Processor issues Dstream Read of address X at BIU port. L1 Victim address is **L**.
2	SLC retimes BIU request, sends address to TAG and DAT arrays
6	Tag array looks up address X. Data array begins data lookup. SLC may send write operations for **L** at any time.
10	Tag is a miss on both ways. Way W is selected as victim. Victim block address is V. Data array muxes data from way W (all 8 words) into the Writeback Buffer.
14	Drive RDV (Dstream) or RDSV (Istream) onto CSW as appropriate. Send probe for block V to processor BIU. Writes from SLC to address **V** are all routed to the WriteBack buffer. Writes from SLC to address **L** are all routed to the L2 data array as a normal L1 write. (See Table 6.3.)
P1	Probe completes in processor – invalidate the block, returns DIRTY if block must be written back. Writeback operations from BIU to L2 data array begin after probe response. SLC retimes writeback data, inserts data into WriteBack buffer. (Overwrites L2 data.) Send probe for block V+32 to processor BIU
P2	Probe completes in processor. If neither block is DIRTY, send WBCANCEL to CSW.
P2+4	Dump WriteBack buffer to CSW to complete RDV or RDSV writeback portion.
T	CSW returns first 16 bytes (DAT[0], DAT[1]) of data to Data array Fill Buffer
T+4	CSW returns DAT[2], DAT[3] to Fill Buffer. SLC retimes DAT[0], DAT[1] to processor BIU. DAT[0], DAT[1] written to data array. (This may be delayed if the L2 data array is busy.) Update TAG array with current MOD STATE.
T+8	SLC retimes DAT[2], DAT[3] to processor BIU. DAT[2], DAT[3] written to data array.
T+12	DAT[4], DAT[5] written to data array.
T+16	DAT[6], DAT[7] written to data array.

Table 6.6: L1 Read Miss, L2 Read Miss with L1 and L2 evictions

6.15.5 Putting It All Together

We're pretty tight for space in the processor segment. In particular, we're limited as to how much room we have for queues and attendent state aside from the 256KB worth of data in the L2 arrays. Figure 6.9 shows the major components of the L2 portion of the processor segment and the total bytes of RAM, buffer, and register storage for each. Earlier sections have described the significant features of the tag and data arrays. The controller is responsible for all command parsing from the CSW and the MIPS BIU, as well as mux control and data steering in the tag and data arrays.

The controller segment also initiates and responds to I/O space accesses (Section 6.18) and interrupts (Section 6.9).

6.15.6 The SLC (slick) and Processor Access Stalls

The SLC is responsible for retiming requests and responses between the PCLK (processor clock) and CCLK (central clock) domains. It also handles all processor stall operations.

While cache fills and victimizations are in progress, we occasionally need to prevent the processor from issuing new requests to the L2 data or tag arrays. There are two levels of stall operation. The first prevents all processor requests and is used in the early stage of a fill or probe operation to allow the CTL unencumbered access to the tag and data arrays. The second level allows write operations to propagate through, but enqueues up to two read operations in the SLC's pending read queue. This is used in the later stage of fill and probe operations to allow invalidate writebacks to wend their way into the DAT array's writeback buffer.

The SLC ARdy state machine that implements "first level" stall and monitors stall requests from the DAT and CTL units. Note that cac_cpu_ARdy_pr and cac_cpu_WDRdy_pr are wired together.

6.16 Initial Program Load and Processor Start-up

The processor segment implements the address request half of the initial program load process described in Section 12.8.

6.17 Memory and IO Ordering Rules and Behavior

Here are the simple rules for ordering behavior from the point of view of the processor and the programmer:

1. To ensure that any memory reference A becomes apparent to other processors or an IO device before some other memory reference B, the programmer must insert a SYNC instruction between A and B. The sequence **READ Mem[X]; WRITE Mem[Y]** may be executed in inverse order if X and Y are not in the same 32 byte L1 block.

2. IO WRITE references will complete in order. The sequence **READ IoSpace[X]; WRITE IoSpace[Y]** may reorder to **WRITE IoSpace[Y]; READ IoSpace[X]** but **WRITE IoSpace[Y]; Read IoSpace[X]** will never reorder. That is, READ operations to IO space will be deferred until all IO and Memory space writes have completed and become apparent to the rest of the ICE9.

3. IO WRITE and IO READ operations to CacLoc registers (the ICR registers, the CAC ECC Control registers, the SPCL register window, and the Interrupt Delivery registers) may re-order with respect to each other and with respect to IO WRITE operations to other parts of the address space. This means that SYNC instructions should be used to guard ordering for all such operations to the local control registers. Memory write operations, however ARE ordered with respect to IO WRITE operations to any of these registers.

4. The ICE9 MIPS processor implements "hits under misses." This means that reads may re-order relative to each other *in the absense of a SYNC or other ordering event.* In particular, no ordering of READs is implied by the code in Figure 6.10 even if a[] and b[] are written by a process that inserts a SYNC between the update of the two. Figure 6.11 shows that the read-order can be enforced by making the second read operation depend on the result of the first. (A SYNC would work too.)

The CAC unit processes IO write operations in order. The CAC also ensures that IO writes won't re-order relative to IO reads. Some may interpret the MIPS ordering rules as requiring a sync between IO writes and subsequent IO reads and vice versa. However, it is clear that many Linux IO drivers take liberties with the

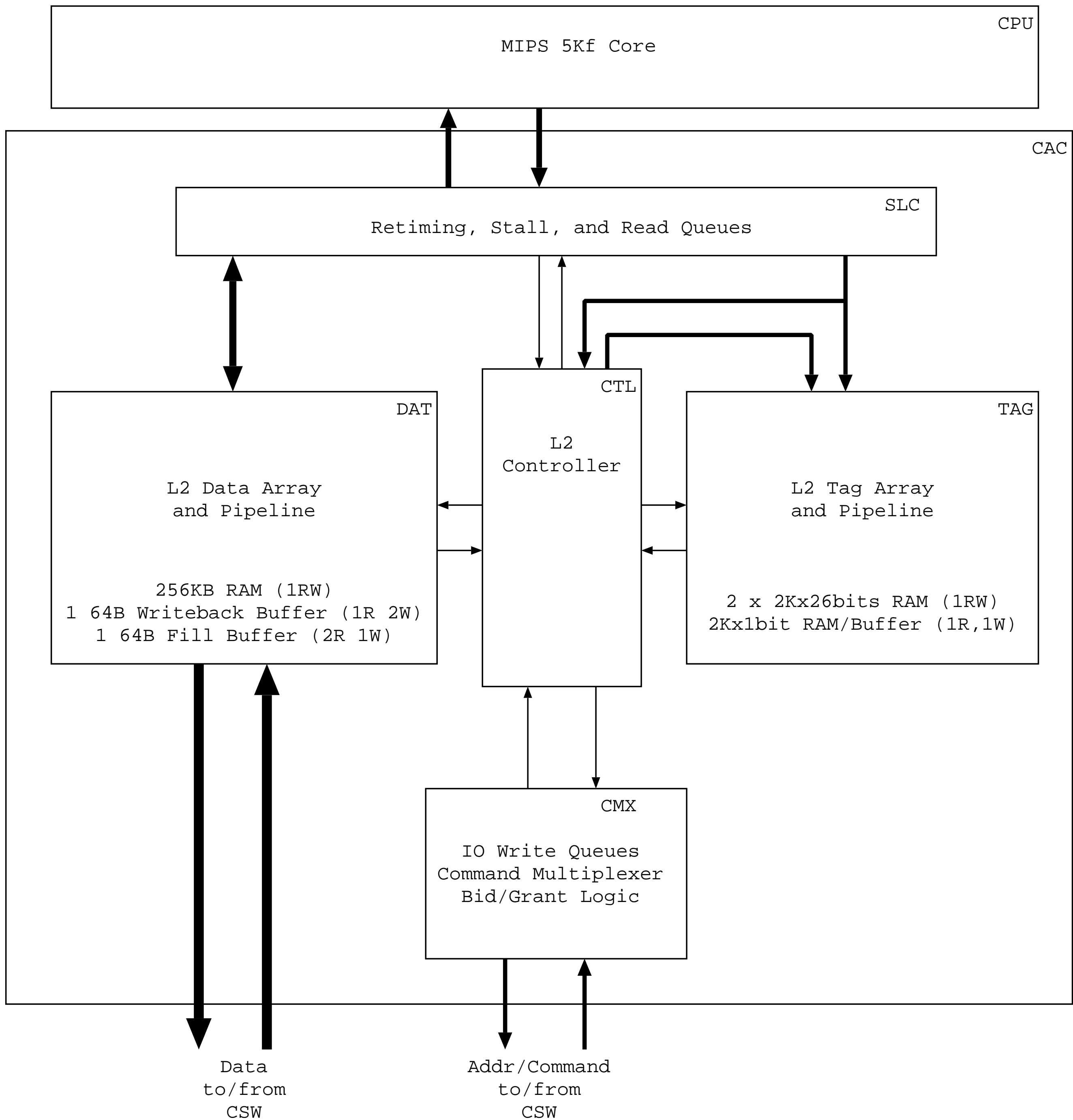


Figure 6.9: Processor Segment L2 Major Units

```
int a[1000], b[1000];
int j, k;
```

Process WRITER

```
b[1] = new_B_value;
SYNC();
a[1] = new_A_value;
```

Process READER

```
j = a[1];
k = b[1]; // j may see new_A_value while k sees old_B_value
```

Figure 6.10: Unordered Reads

```
int a[1000], b[1000];
int j, k;
```

Process WRITER

```
b[1] = new_B_value;
SYNC();
a[1] = new_A_value;
```

Process READER

```
j = a[1];
if(j > 3) k = b[1]; // if j sees new_A_value then k must see new_B_value
```

Figure 6.11: Read Order Enforced by Dependency (assumes no re-ordering of operations by the compiler.)

ordering rule and work better if we can garuntee that IO reads and writes don't pass each other. Therefore, the CAC unit will enqueue all IO reads from the processor and will not pass them on to the CSW until all previously issued IO writes have been completed. An IO write completes when the target device (the device owning the register to which the write is directed) has issued the companion RDIO operation to get the WTIO data. (See Section 6.21.6.)

The MIPS core may emit up to 5 IO writes at a time. The CAC handles only one IO write at a time, so there is a queue in the CMX (command multiplexer) unit that ensures IO writes are completed in order. IO writes may pass L1/L2 writeback operations, but this will not affect the "observed" order of memory updates vs. IO writes, as the cache coherence mechanisms are such that the newly written memory data will be observed by any devices that could observe the newly written IO data.

It should be noted that the CAC enforces IO write ordering and IO write-vs-read ordering as noted above. However, IO writes to the SPCL delivery addresses or to the INT delivery register may pass other IO writes in flight. To ensure that SPCL and INT IO operations do not pass earlier IO transactions, applications should use a SYNC instruction as a barrier before the SPCL or INT op where necessary.

6.18 I/O Accesses and Address Decoding

The CAC unit processes IO read operations in order, and won't reorder them relative to other read operations. IO reads may be reordered relative to L1 to L2 writeback operations as they are processed by the CAC, but the apparent order to all other devices will not violate ICE9 ordering rules. See Section 6.17.

6.18.1 CAC Local IO Registers

There are only a few registers local to the CAC. All are directly accessible only by the local processor. The addresses and register layouts are described in 7.18 on page 444.

6.18.2 CAC Remotely Accessible IO Registers

Currently there are no remotely accessible IO registers other than those provided on the SCB.

6.19 Interrupts, Again

We've talked about interrupts in a number of places. This is the final resting place of all interrupt controversy.

6.19.1 CPU Interrupt lines

Each CPU has 8 interrupts visible to software in the R_CpuCause_IP register. They are defined as follows:

Interrupt	Definition
IP[7]	CPU internal performance counters.
IP[6]	CPU timer interrupts. Internal to each CPU core
IP[5]	Polled, errors and slow devices, or externally vectored by interrupt cause register. (see 7.18.6)
IP[4]	Vectored by interrupt cause register. Kernel assigns for DMA engine.
IP[3]	Vectored by interrupt cause register. Kernel assigns for PCI-Express.
IP[2]	Vectored by interrupt cause register. Kernel assigns for inter-processor interrupts.
IP[1]	Software interrupt. Internal to each CPU core. Asserted and cleared by writing R_CpuCause_IP[1].
IP[0]	Software interrupt. Internal to each CPU core. Asserted and cleared by writing R_CpuCause_IP[0].

6.19.2 The Interrupt Cause Registers

Each PS has a bank of interrupt cause registers, R_CacLocIntCr[7:0]. Each ICR is 64 bits wide and corresponds to one of first four SL_Int level sensitive interrupts: ICR0 and ICR1 to IRQ2, ICR2 and ICR3 to IRQ3, etc. The low 8 bits of the ICR contain the "reason" reported for the corresponding interrupt. Bit 8 of the ICR indicates an "overflow" condition (described below). Bit 9 indicates that the corresponding interrupt is asserted. The remaining bits are read as 0.

When the interrupt handler wishes to dismiss an interrupt, it must write a 1 to bit 9 of the related ICR. This will clear the interrupt cause register and deassert the related interrupt.

Finally, we have the problem of two interrupts arriving to write the same ICR before the first one has been handled and dismissed. In this case, the second interrupt request will set the OVERFLOW bit in the target ICR. No other bits in the ICR are affected by the second request. This means that software must poll all possible origins of requests to a given ICR whenever the overflow bit is set.

The Interrupt Cause Registers are arranged as 64 bit I/O registers in each processor's private address space. (That is, no processor can directly access another processor's ICRs. For an explanation of indirect access, see Section 6.19.4.)

6.19.3 The CSW INT Transaction and Writing the Interrupt Cause Registers

The CSW INT command (see Section 7.10.5) appropriates the address field of the address/command "bus" to carry the interrupt cause and a choice of which interrupt to assert and which ICR to write. Bits 10:8 of the incoming "address" select the ICR from the set of 8 ICRs. Bits 10:9, by implication, select which interrupt will be asserted. Bits 7:0 are written to the appropriate ICR. A processor may deliver an interrupt to itself.

6.19.4 Interprocessor Interrupts

Any processor can send an interrupt to any other processor via the interrupt delivery register. Writes to R_CacLocIdr will cause a CSW INT to the appropriate destination node. The IDR is described in Section 7.18.7. To deliver an interrupt to processor X, a processor Y will write X's bus stop number, the index into X's set of ICRs, and a reason code. The PS interface to the CSW will convert this I/O write to a CSW INT transaction. By convention, interrupt input 0 (IRQ2) is used for inter-processor interrupts.

Note that this mechanism allows any processor to spoof interrupts from any device. That may come in handy some day.

6.19.5 Machine Check Interrupts

This section is obsolete – we have no "machine check" interrupt.

6.19.6 "Slow" Interrupts

Some ICE9 on-chip components need to originate interrupts, but don't have a direct or convenient path to the CSW (where they could originate an INTR command). To accomodate this the OCLA Lac, PMI, SCB, FL, DMA, FSW, UART and two COH units each have an interrupt wire they can tug on to indicate a need for service or the occurence of an error condition.

These interrupt signals are routed through the CSW to each of the six L2 Cac interfaces. Each Cac may select which of the interrupt sources may cause an interrupt to be signaled with the Slow Interrupt Select register. If an interrupt is asserted and it is also selected (enabled) by the R_CacLocSlIntSel register, processor interrupt input 3 (IRQ5) is asserted and remains asserted until the interrupt condition is cleared. (See Section 7.18.8.) The assertion state of each of the incoming interrupts may also be read from the R_CacLocSlIntSel register.

In addition to the slow interrupts from other devices in the ICE9, the R_CacLocSlIntSel register contains two bits indicating the detection of a correctable or uncorrectable ECC error. In the event of an uncorrectable error, the CAC will assert the slow error interrupt to the processor. Correctable ECC errors will be signalled as INT[3]. Both error conditions may be cleared by writing a 1 to the appropriate bit in R_CacLocSlIntSel.

6.19.7 Delivering Interrupts to Other Processors

Each ICE9 processor can deliver an interrupt to the ICR of any other processor via the outbound interrupt delivery register R_CacLocIntDel. See Section 7.18.7. Writes to this register become INT requests on the CSW.

6.20 Error Correction, Detection, Control, and Testing

All data passing over the CSW is protected by ECC, as is all data and tag information in the L2 caches. Uncorrectable errors are signalled by asserting the processor's non-maskable interrupt. Correctable errors are signalled by a slow interrupt. (See 6.19.6, and 7.18.8.)

Each CAC, in its own local IO CSR space, provides five registers for control and monitoring of ECC generation, and detection. They follow the scheme described in 12.4.

6.21 Processor/L2 Transactions – NittyGritty Details

This section outlines the flow of data and sequence of control actions for all of the possible transactions that could take place between the processor and the L2 or I/O system.

Most of the cases enumerated here require a lookup in the L2 tag array. In the case of D-stream accesses, we accelerate the tag lookup by launching a speculative lookup using the address sent to the L1 D-cache. When the BIU state machine sends the actual miss request to the cache segment, we check the BIU address against the last speculative miss address. If they match, we make use of the earlier tag lookup result. Otherwise, we send the BIU address through the tag lookup pipeline. In the descriptions that follow, we lump all this tag-lookup machinery into the notion of "performing an L2 tag lookup" without rehashing the details each time.

All 32 byte fills from the L2 to the L1 are delivered in "best word first" order. Responses to probes are delivered from the processor to the L2 in "word 0 first" order. In all cases, probe addresses sent from the L2 to the processor will set address bits [4:3] equal to 0.

6.21.1 Processor L1 Cache Read Miss

6.21.1.1 I-Stream Read L1 Miss, L2 Hit

I-stream read L1 misses are recognized by the assertion of **cpu_cac_reqAValid_pr**, **cpu_cac_reqBurst_pr**, and **cpu_cac_reqInstr_pr**. (If burst is not asserted, then this I-stream access is bypassing the L1 cache. See Section 6.21.3.)

The TAG unit will signal an L2 hit to the CTL after performing an L2 tag lookup on the BIU request. (The SLC will multiplex the BIU address onto the TAG index and address comparison inputs.) The CTL then directs the DAT unit to perform a 32 byte read of the appropriate block. The DAT sends the 32 byte block to the SLC. The CTL tells the SLC to sequence a burst read back to the processor's **cpu_cac_rtnRData_pr** bus.

6.21.1.2 I-Stream Read L1 Miss, L2 Miss

I-stream read L1 misses are recognized by the assertion of **cpu_cac_reqAValid_pr**, **cpu_cac_reqBurst_pr**, and **cpu_cac_reqInstr_pr**. (If burst is not asserted, then this I-stream access is bypassing the L1 cache. See Section 6.21.3.)

The TAG unit will signal an L2 miss to the CTL after performing an L2 tag lookup on the BIU request. (The SLC will multiplex the BIU address onto the TAG index and address comparison inputs.) The TAG unit also reports the choice of the victim block and its state to CTL.

If the state of the victim block is SHARED, or INVALID the CTL will send a RDS command to the CSW. When the data returns, the CTL will route the data through the DAT unit to the SLC. The SLC will retime the first 32 bytes of the return data onto the **cac_cpu_rtnRData_pr** bus.

If the state of the victim block is EXCLUSIVE, MODIFIED, or UPDATED, The CTL will direct the TAG unit to send a probe request to the processor via the **cac_cpu_prb*** inputs. (See Section 6.21.8.) At the same time, the CTL will send an RDSV command (read shared with victim) to the CSW. When data returns, it will be routed to the BIU in the same manner as for an RDS transaction.

6.21.1.3 D-Stream Read L1 Miss, L2 Hit

D-stream read L1 misses are recognized by the assertion of **cpu_cac_reqAValid_pr**, and **cpu_cac_reqBurst_pr**, and the deassertion of **cpu_cac_reqInstr_pr**. (If burst is not asserted, then this D-stream access is bypassing the L1 cache. See Section 6.21.3.)

The TAG unit will signal an L2 hit to the CTL after performing an L2 tag lookup on the BIU request. (The CTL may make use of the tag lookup performed using the "fast path" described above.) The CTL then directs the DAT unit to perform a 32 byte read of the appropriate block. The DAT sends the 32 byte block to the SLC. The CTL tells the SLC to sequence a burst read back to the processor's **cpu_cac_rtnRData_pr** bus.

6.21.1.4 D-Stream Read L1 Miss, L2 Miss

D-stream read L1 misses are recognized by the assertion of **cpu_cac_reqAValid_pr**, and **cpu_cac_reqBurst_pr**, and the deassertion of **cpu_cac_reqInstr_pr**. (If burst is not asserted, then this D-stream access is bypassing the L1 cache. See Section 6.21.3.)

The TAG unit will signal an L2 miss to the CTL after performing an L2 tag lookup on the BIU request. (The CTL may make use of the tag lookup performed using the "fast path" described above.) The TAG unit also reports the choice of the victim block and its state to CTL.

If the state of the victim block is SHARED, or INVALID the CTL will send a RDEX command to the CSW. When the data returns, the CTL will route the data through the DAT unit to the SLC. The SLC will retime the first 32 bytes of the return data onto the **cac_cpu_rtnRData_pr** bus.

If the state of the victim block is EXCLUSIVE, MODIFIED, or UPDATED, The CTL will direct the TAG unit to send a probe request to the processor via the **cac_cpu_prb*** inputs. (See Section 6.21.8.) At the same time, the CTL will send an RDV command (read shared with victim) to the CSW. When data returns, it will be routed to the BIU in the same manner as for an RDEX transaction.

6.21.2 Processor L1 Cache Write Miss

All L1 misses caused by a store instruction are converted into L1 read miss requests by the BUI. See Section 6.21.1.

6.21.3 Processor L1 Cache Bypass Read to Cacheable Memory

Earlier versions of this specification indicated that the L2 segment would support 64 bit reads (that is, non-burst reads) to memory. This is no longer supported. Such reads to memory produce an unpredictable result. (Such reads can only be caused by certain accesses to non-cached memory space.)

6.21.4 Processor L1 Cache Bypass Write to Cacheable Memory

Earlier versions of this specification indicated that the L2 segment would support 64 bit writes to memory. This is no longer supported. Uncached writes to memory space produce unpredictable results.

6.21.5 Processor I/O Read

Processor read operations to non-cacheable addresses (addresses with the MSB of the physical address set) are passed on to the CSW or to the processor segment's local registers. Such operations are recognized by the assertion of **cpu_cac_reqAValid_pr**, **cpu_cac_reqAddr_pr[35]**, and the deassertion of **cpu_cac_reqWrite_pr**. If **cpu_cac_reqBurst_pr** is asserted, the operation will return 0 for all 32 bytes in the burst.

In the case of local register read operations (the address falls in the range of this PS segment's I/O range, or in the CPULOC I/O range – see Section 16.6.6) the CTL will select the appropriate register and steer its data to the SLC. The SLC will sequence the data onto the BIU data pins. Note that the CTL is responsible for address decoding and sequencing operations. The interrupt reason registers are in the CPULOC I/O range.

I/O accesses that are outside the CPULOC I/O range must be sent to the appropriate device. The CTL selects the device and initiates the CSW RDIO transaction. When data returns, the DAT unit notifies the CTL and the CTL steers the incoming data from the DAT unit to the SLC where it is sequenced onto the BIU. For operations sent to the CSW, the byte enable vector **cpu_cac_reqBE_pr[7:0]** is passed along to the CSW. In general, this is irrelevant to I/O registers created by SiCortex, as we prohibit reads from causing side-effects. However, we don't own all the I/O devices on the chip, so we must provide machinery that honors the size of I/O read requests.

6.21.6 Processor I/O Write

Processor write operations to non-cacheable addresses (addresses with the MSB of the physical address set) are passed on to the CSW or to the processor segment's local registers. Such operations are recognized by the assertion of **cpu_cac_reqAValid_pr**, **cpu_cac_reqAddr_pr[35]**, and **cpu_cac_reqWrite_pr**. If **cpu_cac_reqBurst_pr** is asserted, the operation will return 0 for all 32 bytes in the burst.

In the case of local register read operations (the address falls in the range of this PS segment's I/O range, or in the CPULOC I/O range – see Section 16.6.6) the CTL will select the appropriate register and steer its data to

the SLC. The SLC will sequence the data onto the BIU data pins. Note that the CTL is responsible for address decoding and sequencing operations. The interrupt reason registers are in the CPULOC I/O range.

I/O accesses that are outside the CPULOC I/O range must be sent to the appropriate device. The CTL selects the device and initiates the CSW WTIO transaction, and loads the write data into the **WtIoDat** register in the DAT unit. When the targetted unit responds with the completing **RDIO** transaction, the CTL unit will direct the DAT unit to drive the contents of WtIoDat onto the CSW data bus. When the DAT unit recieves a data grant, the CTL will complete the write operation. Since the processor may queue up to 5 I/O writes for processing, such requests enter an I/O write queue in the SLC and are processed in order. For operations sent to the CSW, the byte enable vector **cpu_cac_reqBE_pr[7:0]** is passed along to the CSW.

Once the I/O write has completed, the CTL unit sends a write buffer acknowledgement back to the processor through the SLC via the **cac_cpu_wbAck_pr** signal.

6.21.7 Processor L1 Eviction

A processor may evict an L1 block at any time. It signals a block writeback with the assertion of **cpu_cac_reqAValid_pr**, **cpu_cac_reqWrite_pr**, and **cpu_cac_reqBurst_pr**. Block writes to I/O space are ignored.

When the L1 eviction address passes from the SLC to the CTL, the CTL will setup the DAT pipeline for an L1 writeback. When the pipeline is ready, the CTL will tell the SLC to assert **cac_cpu_WDRdy_pr**. The processor then sources data onto the **cpu_cac_WData_pr** bus which is retimed by the SLC unit to 128 bits. The DAT pipeline appends ECC bits and writes the 32 byte block into the L2 data array. The Tag unit changes the state of the block to “UPDATED.”

6.21.8 L2 Probe to Processor

The L2 cache may initiate a probe request to the processor for two reasons. First, the L2 may need to displace a block due to a L2 miss caused by a processor request. (See Section 6.21.1.) Second, the L2 may launch a probe into the L1 to retrieve a potentially dirty block in response to a CSW PRBWIN command. In either case, the actions by the L2 segment and the processor are identical. Note that in both cases, the CTL will send in TWO probe requests, one for each of the two 32 byte blocks that map to the 64 byte block of interest. They will be sent in order and the second will not be sent until after the first probe has been acknowledged via the **cpu_cac_invAck_pr** signal.

The probe is launched by the CTL. CTL sends a probe request to the SLC unit which drives the probe address onto **cac_cpu_prbAddr_pr**, and asserts **cac_cpu_prbReq_pr**. At some later time, the processor will respond with some combination of assertions of **cpu_cac_invHit_pr**, **cpu_cac_invDirty_pr**, and **cpu_cac_invLock_pr** along with the assertion of **cpu_cac_invAck_pr**.

6.21.8.1 Probe Hits on Clean Block

When the probe hits on a clean block, the processor will assert **cpu_cac_invAck_pr** while deasserting **cpu_cac_invDirty_pr**. In this case, the CTL will not anticipate a writeback for the block. The CTL will release the DAT data path to send probe or victim data out to the coherence widget or probe requester. If both probes for a block return clean and the block is clean in the L2, the CTL will initiate a WBCANCEL operation in the event of a writeback, or complete the PRBINV operation.

6.21.8.2 Probe Hits on Dirty Block

When the probe hits on a dirty block, the processor will assert **cpu_cac_invAck_pr** and **cpu_cac_invDirty_pr**. In this case, the CTL expects a writeback for the block. Before launching the probe, the CTL pre-arms the DAT unit to prepare it to accept writeback data. With the arrival of invDirty, the SLC alerts the DAT unit that writeback data is arriving. Once both probes have completed, the CTL unit will complete the writeback or probe operation by signalling the DAT unit to forward the updated data to the CSW.

6.21.8.3 Probe Misses in L1

When the probe misses in the L1 D-cache, the processor will assert **cpu_cac_invAck_pr** while deasserting **cpu_cac_invDirty_pr**. (This is indistinguishable from a hit on a clean block. The **cpu_cac_invHit_pr** signal may be used as a hint for performance counters, but is not to be used in making protocol decisions.) In this case, the CTL will not anticipate a writeback for the block. The CTL will release the DAT data path to send probe or victim

data out to the coherence widget or probe requester. If both probes for a block return clean and the block is clean in the L2, the CTL will initiate a WBCANCEL operation in the event of a writeback, or complete the PRBINV operation.

6.22 L2 Responses to Probe Requests

In the responses described below, data is returned – if required – to the appropriate requester via the DAT unit. The DAT unit is responsible for sequencing data responses to the CSW and waiting for data grant.

Note that the cache segment stalls all new read accesses from the processor while probe handling occurs. This avoids ships-passing-in-the-night problems with, for instance, a PRBBWT arriving, finding a hit, and then finding that the target block has been evicted when the BWT data arrives. Further, when the cache segment responds to a PRBBWT with a BWTGO, it will wait until all outstanding L2 fills or IO reads have completed, since the CSW port can handle just one block of data coming to a processor in any given cycle.

All probe handling begins with the dispatch of the operation at the top of the CTL probe state machine. When an enqueued probe request is found, the CTL unit causes the SLC to "pause" the processor BIU. Each probe flow will wait for the SLC to signal that the BIU is now in the paused state. (The BIU is paused when ARdy is deasserted and there are L1 instigated operations currently in flight.) Figure 6.12 shows the probe operation dispatch.

6.22.1 PRBINV

The CTL unit will drive the incoming address to the Tag unit. At the same time, the CTL will direct the SLC to hold off further processor BIU requests (via the **cac_cpu_reqARdy_pr** signal) while the Tag array is occupied. The Tag unit will use **ctl_xxx_Addr_c2a** to generate a lookup and set the matching state (if any) to INVALID unless the current state is EXCL/DIRTY/UPDATED.[1] Incoming PRBINV commands that carry a TID that is owned by the receiving unit do not update the cache, but send an INVDONE to the originating COH unit.

The CTL generates an INVDONE response to the CSW when a PRBINV has been handled (or when a PRBINV for a TID owned by this unit arrives). The CTL state machine flow for PRBINV is shown in Figure 6.13.

6.22.2 PRBWIN

The CTL unit will drive the incoming address to the Tag unit. At the same time, the CTL will direct the SLC to hold off further processor BIU requests (via the **cac_cpu_reqARdy_pr** signal) while the Tag array is occupied. The Tag unit will use **ctl_xxx_Addr_c2a** to generate a lookup and set the matching state (if any) to INVALID. The Tag unit will also report the result of the lookup to the CTL.

If the lookup missed in the Tag, the CTL unit will initiate a PRBNOHIT response to the original requester.

If the lookup hit in the Tag the CTL will initiate two probes into the L1 cache for the two L1 blocks contained in the identified L2 block. After the L1 probes complete (see Section 6.21.8.) the CTL will direct the DAT unit to send the L2 block to the requester.

(Note that if the L2 block was in the SHARED or INV state, we still do the probes. PRBWIN shouldn't arrive for blocks in the SHARED state, but as the L2 ignores the state in this case, we'll complete the transition. However, the hardware is broken at this point.)

The BRD, WIN, and SHR flows all share a commong writeback flow shown in Figure 6.16.

6.22.3 PRBBRD

The CTL will drive the incoming address to the Tag unit. At the same time, it will launch two probe requests to the L1.

After the probe requests complete, and any data has been written to the L2, the CTL directs the L2 to respond with data as for a PRBWIN request. In this case, however, the tag array state remains unchanged. The transaction is handled by the CTL PRB state machine as shown in Figure 6.15

[1] In this case, the PRBINV is "stale" and arrived sometime in the past, was neglected until now, and is now applied to a block that was recently acquired.

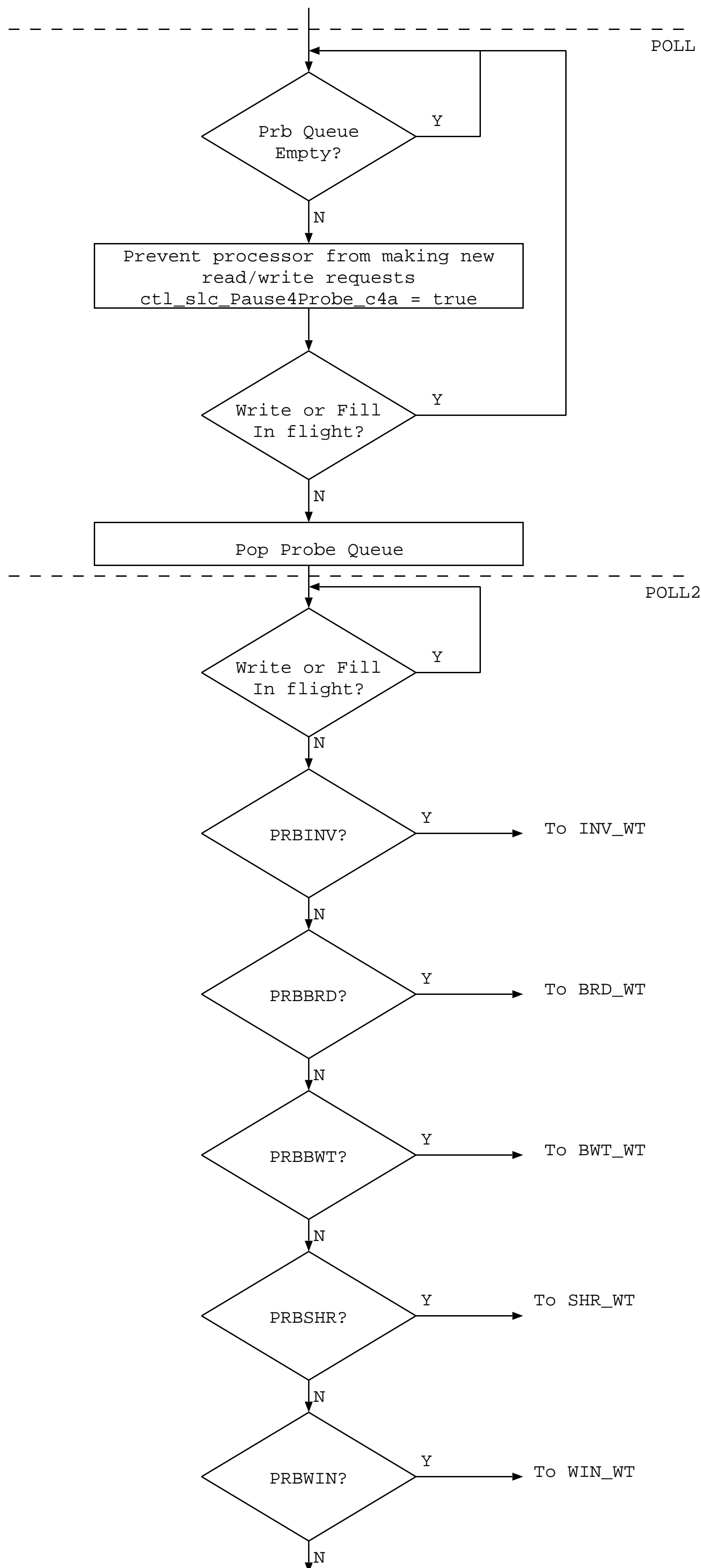


Figure 6.12: Probe Operation Dispatch

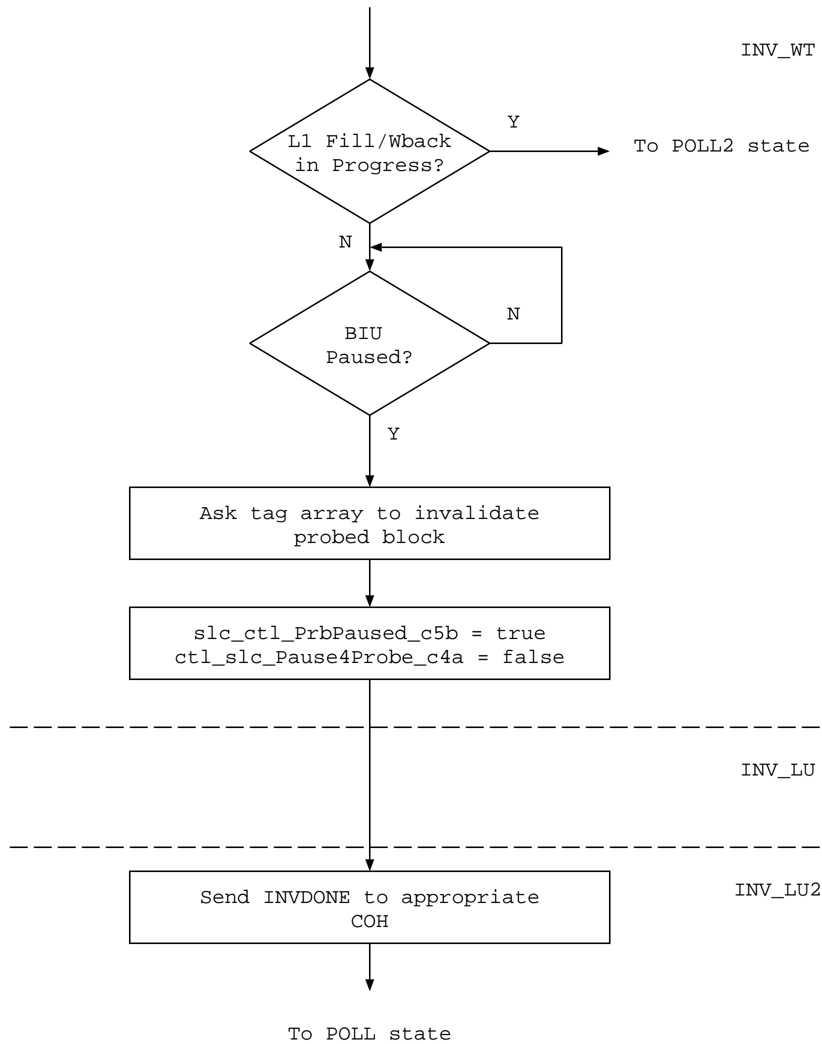


Figure 6.13: CTL State Machine Flow for PRBINV

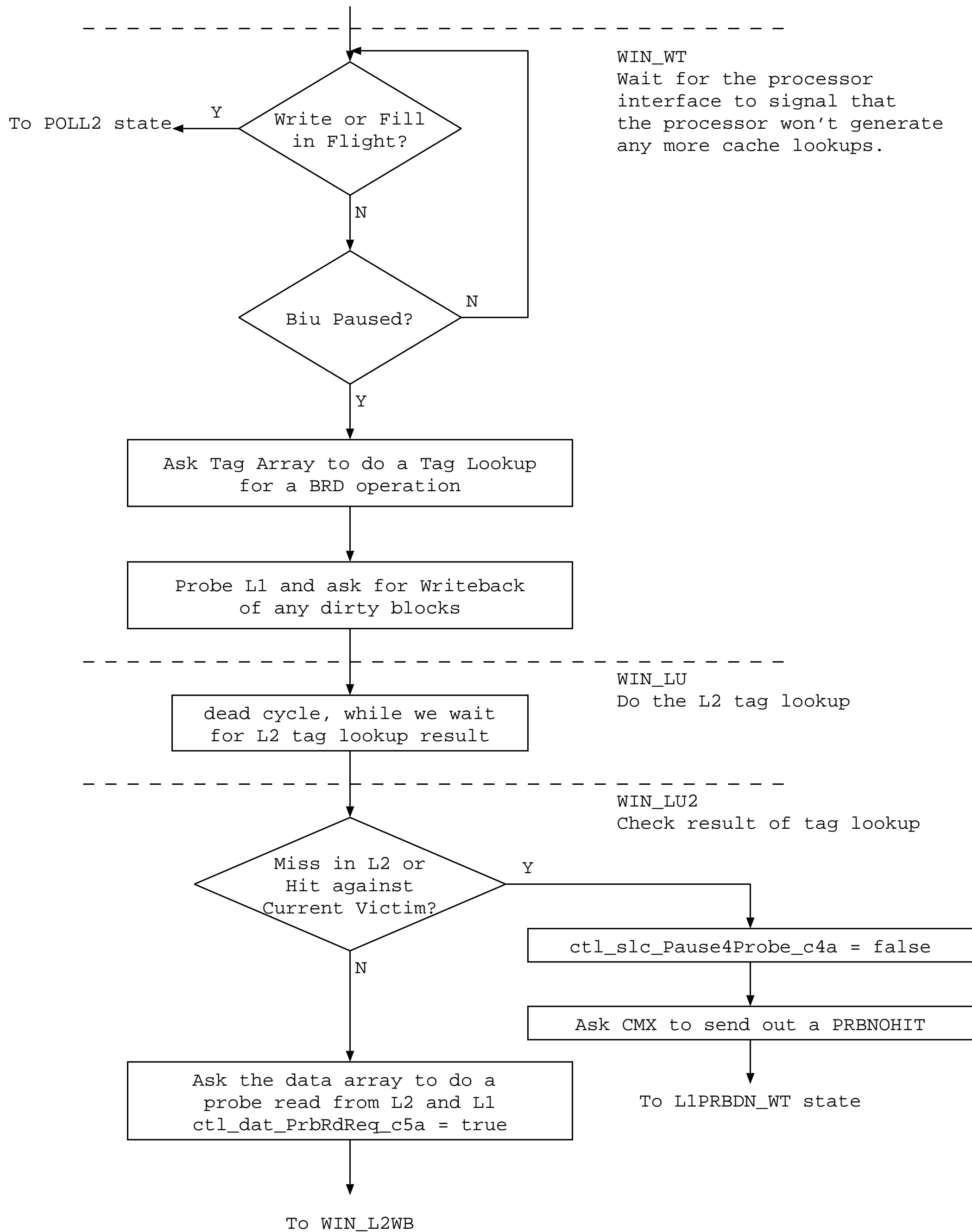


Figure 6.14: CTL State Machine Flow for PRBWIN

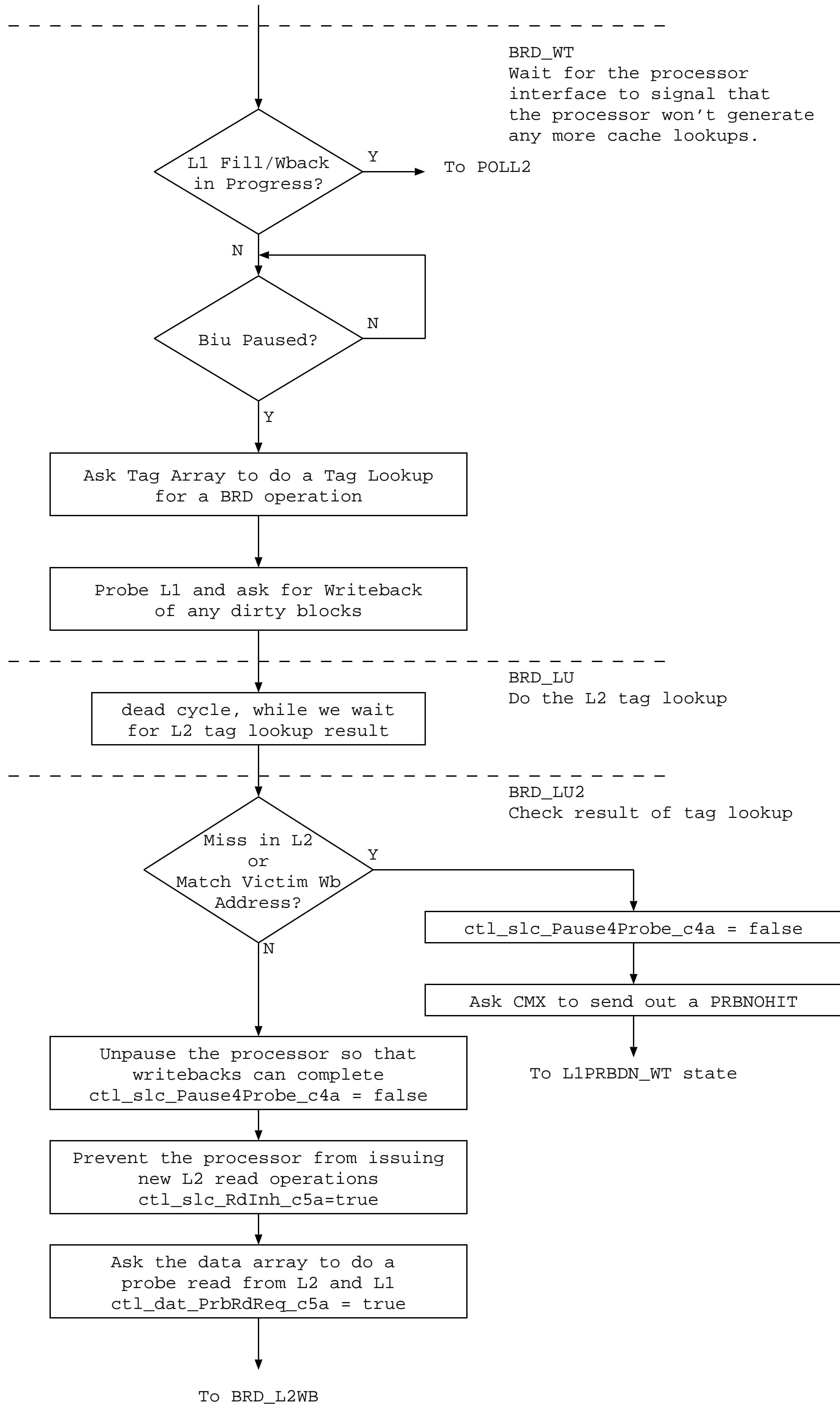


Figure 6.15: CTL State Machine Flow for PRBBRD

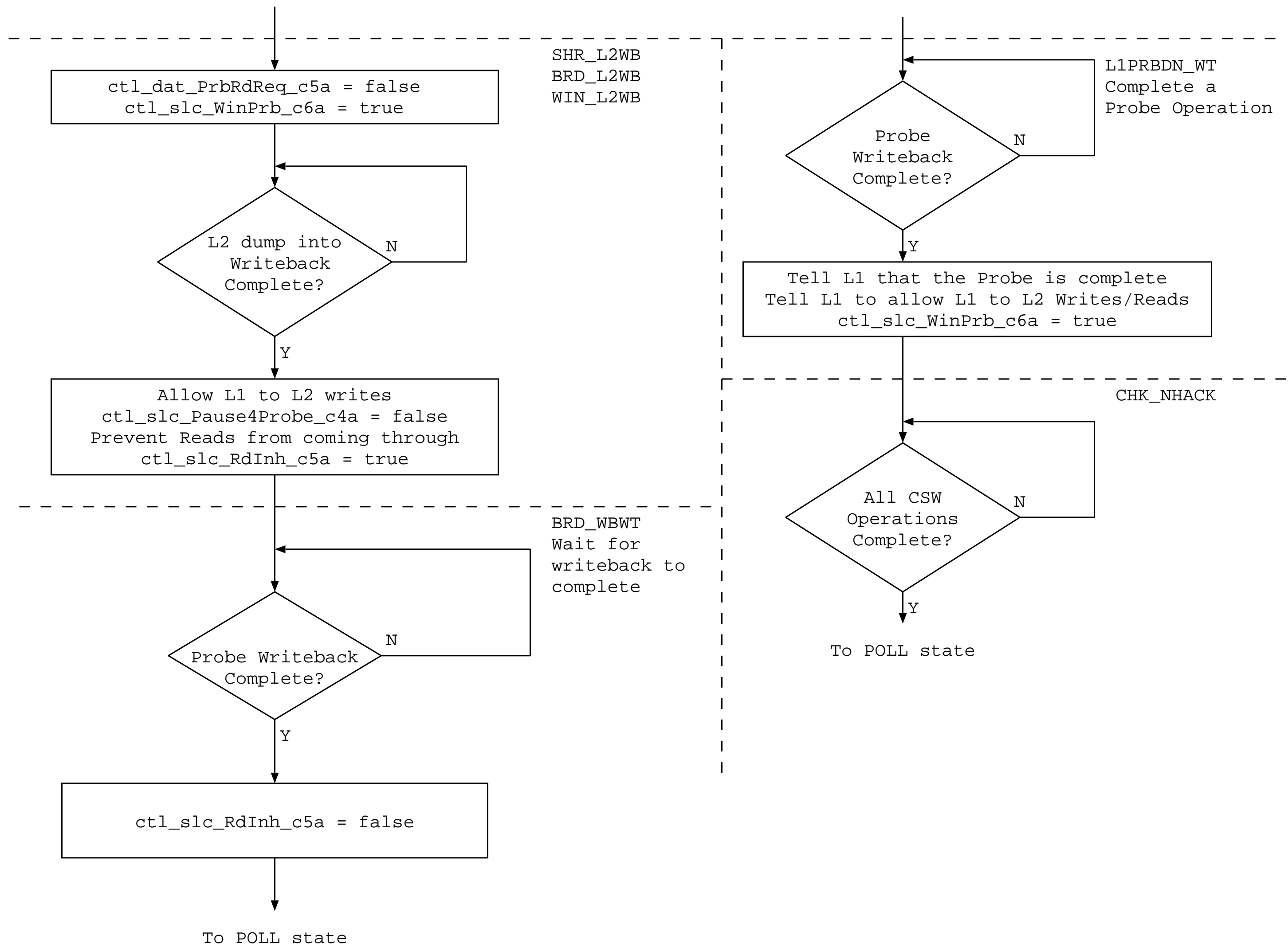


Figure 6.16: Common Writeback Flow

6.22.4 PRBBWT

The CTL will drive the incoming address to the Tag unit. At the same time, it will launch two probe requests to the L1. The CTL will direct the SLC and DAT units to ignore the return data (if any) from the L1. (The object here is to clear the valid bits for the relevant blocks in the L1 cache._

After the probe requests are launched (but we don't wait for acknowledgement) the CTL originates a BWTGO command on the CSW to prompt the requester to send the block of data.

When the data arrives, the DAT unit will write the data into the L2 data array. During this operation, the CTL will direct the SLC to hold off all BIU write data with the **cac_cpu_reqWDRdy_pr** signal.

After the DAT unit signals to the CTL that the transfer has completed, the CTL will send a BWTDONE signal to the COH. (This allows the coherence engine to "complete" the write action and t rigger any operations that were dependent on the block write.

Note that only the DMA engine and the PCI-e controller can initiate BWT operations, so the PS need only provide enough book-keeping slots to keep track of 8 BWT operations at a time. They will always arrive with a TID of DMAWTx or PCIWTx where x can range from 0 to 3 inclusive.

6.22.5 PRBSHR

A PRBSHR request will arrive when this processor has cached a a block in any state and another processor also wishes to cache the block in SHARED state. Figure 6.18 shows the flow.

6.23 L2 Responses to Other CSW Commands

6.23.1 PRBNOHIT

PRBNOHIT arrives in response to a forwarded RDEX, RDV, RDS, or RDSV operation. In this case, the CSW immediately drives the appropriate retry (RDEXR or RDSR) operation onto the CSW.

6.23.2 RDIO

The CTL unit will check the incoming csw address against the known address ranges fielded by this node. CTL will drive the incoming address **ctl_xxx_Addr_c2a** and assert **ctl_xxx_IORd_c2a**.

If the address is out of bounds (*i.e.* does not match any range), the CTL will direct the DAT unit to initiate a data response with a data field of all zeros.

If the address is in bounds, the CTL will drive a unit select signal to the PS unit that owns the registers. The target unit will use the **ctl_xxx_Addr_c2a** signal to select the appropriate register and drive return data to the DAT unit. The DAT unit will pass the data on to the CSW.

6.23.3 WTIO

The CTL unit will check the incoming csw address against the known address ranges fielded by this node. CTL will drive the incoming address **ctl_xxx_Addr_c2a** and assert **ctl_xxx_IOWt_c2a**.

As noted in the L2 Cache chapter, WTIO transactions are double-ended so as to allow the processor node to prevent two data blocks from arriving at a CSW bus stop at the same time. On receipt of an WTIO, the CTL unit will initiate an RDIO command to the requester using the same TID **Ty** as the incoming request. The address attached to the RDIO is "WTIOADDR". At some later time, the data will return from the original requester. The TID will be matched up against **Ty** and the data routed to the appropriate destination.

If the address is out of bounds (*i.e.* does not match any range), the CTL will direct the DAT unit to drop the data into the bit bucket.

If the address is in bounds, the CTL will drive a unit select signal to the PS unit that owns the registers. The target unit will use the **ctl_xxx_Addr_c2a** signal to select the appropriate register and write the incoming data from the DAT unit (on **dat_xxx_IOWtDat_c4a**) into the target register.

WTIO operations may ***not*** be initiated by either the PCIexpress controller or the DMA unit. All WTIO operations are initiated by processors. This is important, as the CAC tracks just one WTIO transaction for each of the processors. (No processor/L2 complex can have more than one write operation outstanding at a time. Other I/O writes from the processor are enqueued in the SLC until resources are available in the CAC to handle them.)

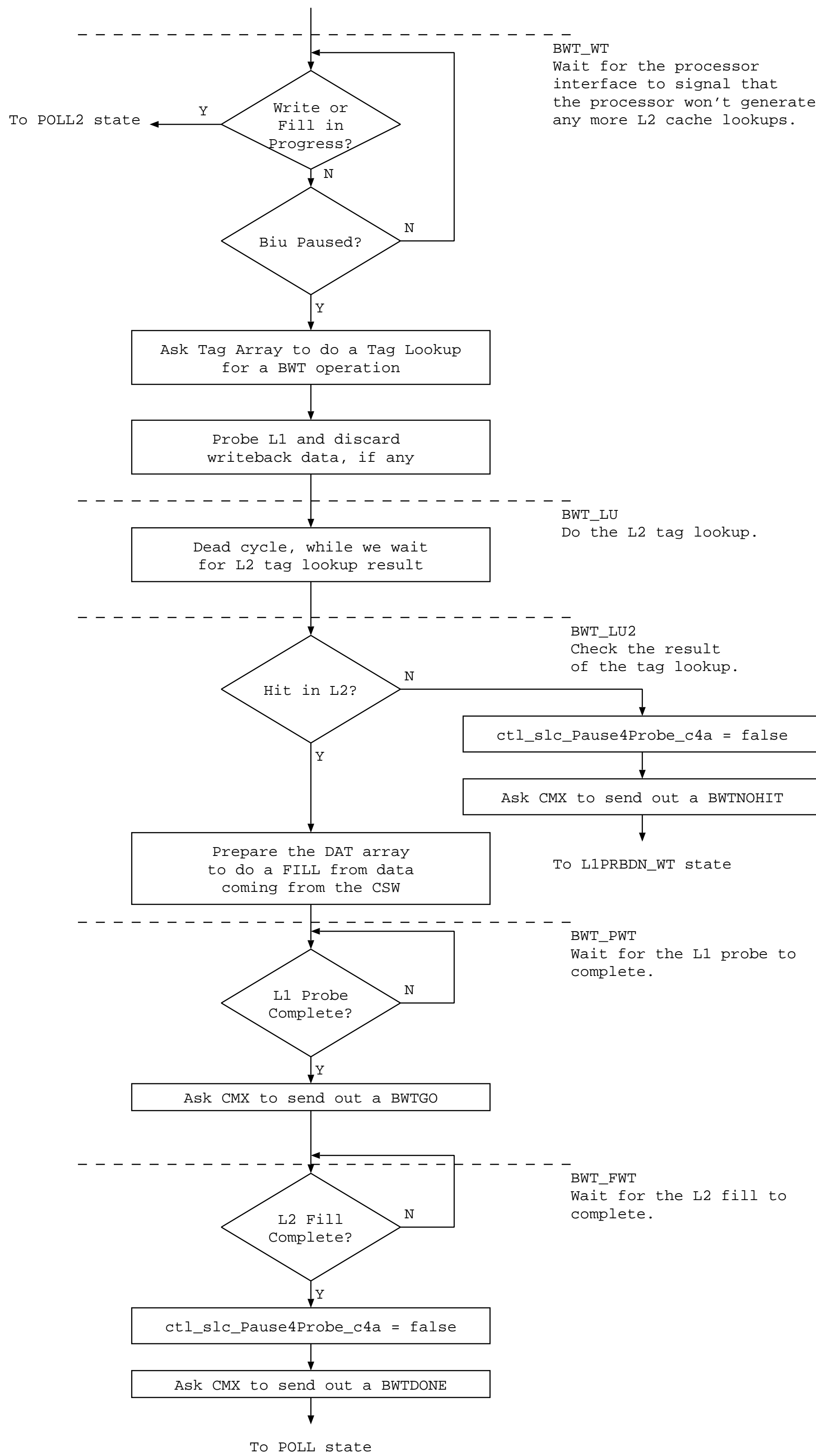


Figure 6.17: CTL State Machine Flow for PRBBWT

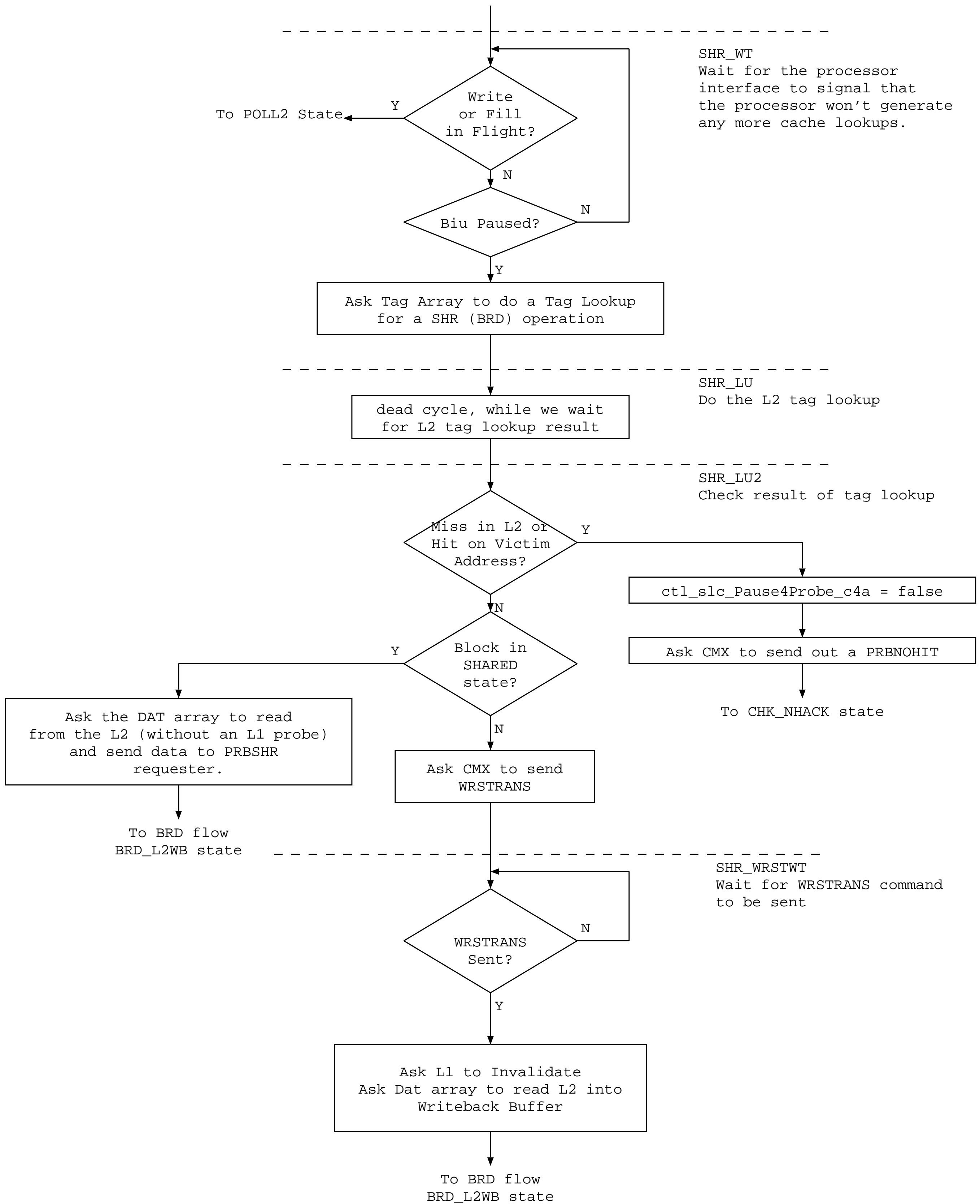


Figure 6.18: CTL State Machine Flow for PRBSHR

6.23.4 INT

See Section 6.19.
On receipt of an INT command, the CTL unit will send a DONE response via the CSW to the originating node.

6.23.5 Incoming Data Completing a Memory Read Operation

When data arrives from the CSW in response to a RDEX, RDV, RDS, or RDSV operation, the CTL unit will check the DataOrigin. If DataOrigin is not a coherence widget, the CTL unit will send a PRBDONE request to the appropriate coherence widget to complete the transaction.

6.24 Registers and Definitions

For details on most of these registers, consult the MIPS 5kf Processor Core Family Software User's Manual.

For a whole host of reasons all registers in the L2 cache portion of the processor segment are defined in the CSW and Coherence chapter. See Section 7.17.

The following CPU registers have been modified relative to the m5kf programmer's manual. The changes are bolded in the register descriptions referenced below.

Register/Field Name	See
R_CpuConfig_LLTIME	6.24.8 on page 327
R_CpuConfig2	6.24.10 on page 327
R_CpuPRId	6.24.56 on page 337
R_CpuErrCtl	6.24.22 on page 330
R_CpuCacheErr	6.24.23 on page 331
R_CpuPerfPEA	6.24.20 on page 330
R_CpuPerfVPC	6.24.19 on page 330

6.24.1 Package Attributes

6.24.1.1 Package

chip_cpu_spec

Attributes

-public_rdwr_accessors

6.24.2 Definitions

Defines

CPU

Constant	Mnemonic	Definition
32'd5	C_LINE_LOG2	Caches line size in power-of-2 bytes. (32 bytes.)
32'd32	C_LINE	Caches line size.
32'd4	DC_ASSOC	DCache Associativity.
32'd15	DC_SIZE_LOG2	DCache Size in power-of-2 bytes. (32KB)
32'h8000	DC_SIZE	DCache Size
32'd4	IC_ASSOC	ICache Associativity.
32'd15	IC_SIZE_LOG2	ICache Size in power-of-2. (32KB).
32'h8000	IC_SIZE	ICache Size
4'd6	WB_ENTRIES	Write Buffer entries.
32'd7	TAGHASH0_LO	Low bit of tag hash field
32'd10	TAGHASH_WIDTH	How many bits in the tag hash index field
32'd17	TAGHASH1_LO	Low bits of the other tag hash field (HASH0 XOR HASH1 -> TagIndex)
32'd19	TAG_WIDTH	How many bits in the tag itself?

6.24.3 Register List

The following enumeration summarizes the CPU registers. Note the constant used includes the register number and select field, the CP0_REG macro may be used to split them up.

Enum

CpuCp0

Attributes

-allowlc

Constant	Mnemonic	Product	Definition
8'o00_0	Index		Index into the TLB entry
8'o01_0	Random		Randomly generated index into the TLB array
8'o02_0	EntryLo0		Low-order portion of the TLB entry for even-numbered virtual pages
8'o03_0	EntryLo1		Low-order portion of the TLB entry for odd-numbered virtual pages
8'o04_0	Context		Pointer to page table entry in memory
8'o05_0	PageMask		Control for variable page size in TLB entries
8'o06_0	Wired		Controls the number of xed (wired) TLB entries
8'o10_0	BadVAddr		Reports the address for the most recent address-related exception
8'o11_0	Count		Processor cycle count
8'o12_0	EntryHi		High-order portion of the TLB entry
8'o13_0	Compare		Timer interrupt control
8'o14_0	Status		Processor status and control
8'o15_0	Cause		Cause of last general exception
8'o16_0	EPC		Program counter at last exception
8'o17_0	PRId		Processor identification and revision
8'o20_0	Config		Configuration register
8'o20_1	Config1		Configuration register 1
8'o20_2	Config2	twc9a+	Configuration register 2
8'o21_0			Reserved
8'o22_0	WatchLo		Low-order watchpoint address
8'o23_0	WatchHi		High-order watchpoint address
8'o24_0	XContext		Extended-addressing page table context
8'o25_0			Reserved
8'o26_0	PerfVPC		Performance virtual program counter address
8'o26_1	PerfVPC1		Performance virtual program counter address
8'o26_2	PerfPEA		Performance physical effective address
8'o26_3	PerfPEA1		Performance physical effective address
8'o27_0	Debug		Debug register
8'o30_0	DEPC		Program counter at exception entering Debug Mode
8'o31_0	PerfCnt		Performance counter interface
8'o31_1	PerfCnt1		Performance counter interface
8'o31_2	PerfCnt2		Performance counter interface
8'o31_3	PerfCnt3		Performance counter interface
8'o32_0	ErrCtl		Parity/ECC error control and status
8'o33_0	CacheErr		Cache parity error control and status
8'o34_0	TagLo		Low-order portion of cache tag interface
8'o34_1	DataLo		Low-order portion of cache data interface
8'o35_0	TagHi		High-order portion of cache tag interface
8'o35_1	DataHi		High-order portion of cache data interface

8'o36_0	ErrorEPC		Program counter at last error
8'o37_0	DSAVE		Debug Exception Save Register
(else)			Reserved.

6.24.4 Prefetch Hint Encodings

The following enumeration is used for the hint field of the pref instruction.

Enum

CpuPrefHint

Constant	Mnemonic	Product	Definition
5'd0	LOAD	TWC9A+	Load Prefetch. Data is expected to be read and not modified. If the address translates and misses in L1, read the data exclusive into the L1.
5'd1	STORE	TWC9A+	Store Prefetch. Data is expected to be written. Implemented same as LOAD.
5'd4	LOADSTR	TWC9A+	Load Streamed. Data is expected to be read once and does not need to be cached. Implemented same as LOAD.
5'd5	STORESTR	TWC9A+	Store Streamed. Data is expected to be written once and does not need to be cached. Implemented same as LOAD.
5'd6	LOADRET	TWC9A+	Load Retained. Data is expected to be read many times, versus the LOADSTR stream. Implemented same as LOAD.
5'd7	STORERET	TWC9A+	Load Retained. Data is expected to be written many times, versus the STORESTR stream. Implemented same as LOAD.
5'd25	NUDGE	TWC9A+	Writeback Invalidate. Data is not to be used, if dirty writeback to L2; if clean, invalidate.
5'd26	LOADL2	TWC9A+	Load Prefetch to L2. Data is expected to be read, but prepare only L2 cache. If the address translates and misses in L1, read the data exclusive into the L2 cache, do not change L1 cache.
5'd27	STOREL2	TWC9A+	Store Prefetch to L2. Data is expected to be written, but prepare only L2 cache. Implemented same as LOADL2.
5'd30	PREPSTORE	TWC9A+	Prepare for Store. Data will overwrite an entire cache line, the data can be filled with zeros. Not implemented, becomes NOP.
(else)		TWC9A+	Reserved. Unimplemented in the core and will be NOPed; note some unimplemented encodings are remain architecturally defined.

6.24.5 CPU Performance Counter Events

The following events are trackable by the CPU performance counters in CP0 register 25. These are different event encodings from the SCB performance counters.

ICE9A uses the same encodings as the M5KF core, which unfortunately has different events for each counter. CpuCntEvent0 are the encoding for ICE9A's counter 0, CpuCntEvent1 is for ICE9A's counter 1.

ICE9B uses a different enumeration from ICE9A, CpuCntEvents, but sanitized it by applying the same enumeration to both counters and added additional events.

Enum

CpuCntEvent0

Attributes

-descfunc

Constant	Mnemonic	Product	Definition **(For more details, see descriptions in CpuCntEvents.)**
6'h00	CYCLES	ICE9A	Cycles.
6'h01	INSFETCH	ICE9A	Instructions fetched.
6'h02	LOAD	ICE9A	Load/pref/sync/cache ops.
6'h03	STORE	ICE9A	Stores.
6'h04	SC	ICE9A	Conditional stores.
6'h05	SCFAIL	ICE9A	Conditional stores that fail.
6'h06	BRANCH	ICE9A	Branches executed.
6'h07	ITLBMISS	ICE9A	ITLB misses.
6'h08	DTLBMISS	ICE9A	DTLB misses.
6'h09	ICMISS	ICE9A	I-Cache misses.
6'h0a	INSSCHED	ICE9A	Instructions scheduled.
6'h0b-6'h0d		ICE9A	Reserved
6'h0e	INSDUAL	ICE9A	Dual issued instructions.
6'h0f	INSEXEC	ICE9A	Instructions executed bit 0.
6'h1f-6'h3f		ICE9A	Reserved.

Enum

CpuCntEvent1

Attributes

-descfunc

Constant	Mnemonic	Product	Definition **(For more details, see descriptions in CpuCntEvents.)**
6'h00	CYCLES	ICE9A	Cycles.
6'h01	INSEXEC	ICE9A	Instructions executed.
6'h02	LOAD	ICE9A	Load/pref/sync/cache ops.
6'h03	STORE	ICE9A	Stores.
6'h04	SC	ICE9A	Conditional stores.
6'h05	FLOAT	ICE9A	Floating point instructions executed. Includes all COP1 instructions, including loads and stores.
6'h06	DCEVICT	ICE9A	Data cache line evicted from L1.
6'h07	TLBTRAP	ICE9A	TLB miss exception traps.
6'h08	MISPRED	ICE9A	Branches mispredicted.
6'h09	DCMISS	ICE9A	Data cache misses.
6'h0a	MSTALL	ICE9A	Scheduling conflict M-stage stalls.
6'h0b-6'h0e		ICE9A	Reserved
6'h0f	COP2	ICE9A	COP2 instructions executed.
6'h1f-6'h3f		ICE9A	Reserved.

Enum

CpuCntEvents

Attributes

-descfunc

Constant	Mnemonic	Product	Definition

6'h00	CYCLES	ICE9B+	Cycles. Incremented by one each processor clock cycle.
6'h01	INSFETCH	ICE9B+	Instructions fetched. Incremented by the number of instructions (0,1,2) fetched by the instruction buffer.
6'h02	LOAD	ICE9B+	Load/pref/sync/cache ops. Incremented by one each time a load, pref, sync, or cache instrucion is executed.
6'h03	STORE	ICE9B+	Stores. Incremented by one each time a store instruction completes M stage, irregardless of if it has completed storing to memory. Note that a store conditional is considered executed even if it fails to perform the store due to the LL bit being clear.
6'h04	SC	ICE9B+	Conditional stores. Incremented by one each time a store conditional, passing or failing, completes M stage.
6'h05	SCFAIL	ICE9B+	Conditional stores that fail. Incremented by one each time a store conditional fails.
6'h06	BRANCH	ICE9B+	Branches executed. Incremented by one each time a conditional branch instruction is executed.
6'h07	ITLBMISS	ICE9B+	ITLB misses. Incremented by each miss in the ITLB.
6'h08	DTLBMISS	ICE9B+	DTLB misses. Incremented by each miss in the DTLB.
6'h09	ICMISS	ICE9B+	I-Cache misses. Incremented by each miss in the I-Cache.
6'h0a	INSSCHED	ICE9B+	Instructions scheduled. Incremented by one each time an instruction is scheduled.
6'h0b	MISPRED	ICE9B+	Branches mispredicted. Incremented by one each time a conditional branch is mispredicted.
6'h0c	FLOAT	ICE9B+	Floating point instructions executed. Includes all COP1 and COP1X instructions, including floating point loads, floating point stores, and floating point conditional branches.
6'h0d	COP2	ICE9B+	COP2 and COP2X instructions executed. Includes all COP2 and COP2X instructions, including COP2 loads, COP2 stores, and COP2 branches.
6'h0e	INSDUAL	ICE9B+	Dual issued instructions. Incremented by *two* each time an instruction pair is dual issued.
6'h0f	INSEXEC	ICE9B+	Instructions executed. Incremented by the number of instructions (0,1,2) which have completed their execution in the integer and floating point units. For this count, an instruction is completed if it has passed its M stage without being killed, or was a SYSCALL, BREAK, SDBBP, or trap. A load instruction is considered as executed if it compeleted the M stage, even though it may not have returned data. MDU and floating point instructions are also counted as completed when they finish M stage, though they may require additional cycles.
6'h10	DCEVICT	ICE9B+	Data cache line evicted. Incremented by one each time a 32-byte line is evicted from the L1 data cache. This includes evictions caused by probes.
6'h11	TLBTRAP	ICE9B+	TLB miss exception traps. Incremented by one on each TLB miss exception trap.
6'h12	DCMISS	ICE9B+	Data cache misses. Incremented by one on each L1 Data cache miss.
6'h13	MSTALL	ICE9B+	Scheduling conflict M-stage stalls. Incremented each cycle the M-stage pipeline is stalled due to scheduling conflicts.
6'h14	L2REQ	ICE9B+	Cachable L2 Cache requests. The count increments when the load completes, which may be many instructions after the load if there are no load data-dependancies.

6'h15	L2MISS	ICE9B+	Cachable L2 Cache requests that miss in local L2. The count increments when the load completes, which may be many instructions after the load if there are no load data-dependancies.
6'h16	L2MISSALL	ICE9B+	Cachable L2 Cache requests that miss in all caches and fill from memory. The count increments when the load completes, which may be many instructions after the load if there are no load data-dependancies.
6'h17	FPARITH	ICE9B+	Floating point arithmetic instructions. Increments for each MADD/ MNADD/ MSUB/ NMSUB/ ADD/ SUB/ MUL/ DIV/ SQRT/ RECIP/ RSQRT.
6'h18	FPMADD	ICE9B+	Floating point multiply-add instructions. Increments for each paired instruction; MADD/ MNADD/ MSUB/ NM-SUB.
(else)		ICE9B+	Reserved.

6.24.6 SCB Performance Core Events

The following CPU counter events are trackable by SCB statistical event counting. This table is inserted twice; one for each counter, into CpuScbEvent under the mnemonics C0_ and C1_. For more details on each event, see the descriptions in CpuCntEvents.

Enum

CpuScbCoreEvent

Constant	Mnemonic	Product	Definition **(For more details, see descriptions in CpuCntEvents.)**
5'h00	CYCLES	ICE9B+	Cycles.
5'h01	INSFETCH_B0	ICE9B+	Instructions fetched bit 0.
5'h02	INSFETCH_B1	ICE9B+	Instructions fetched bit 1. Multiply by 2 and add bit 0 counter for total number of instructions.
5'h03	INSSCHED	ICE9B+	Instructions scheduled.
5'h04	INSDUAL_B1	ICE9B+	Dual issued instructions bit 1. Multiply by 2 and add bit 0 counter for total number of instructions. (Scaled in driver software, so read P0_INSDUAL instead.)
5'h05	INSEXEC_B0	ICE9B+	Instructions executed bit 0.
5'h06	INSEXEC_B1	ICE9B+	Instructions executed bit 1. Multiply by 2 and add bit 0 counter for total number of instructions. (Scaled in driver software, so read P0_INSEXEC instead.)
5'h07	LOAD	ICE9B+	Load/pref/sync/cache ops.
5'h08	STORE	ICE9B+	Stores.
5'h09	SC	ICE9B+	Conditional stores.
5'h0a	SCFAIL	ICE9B+	Conditional stores that fail.
5'h0b	BRANCH	ICE9B+	Branches executed.
5'h0c	ICMISS	ICE9B+	I-Cache misses.
5'h0d	ITLBMISS	ICE9B+	ITLB misses.
5'h0e	DTLBMISS	ICE9B+	DTLB misses.
5'h0f	MISPRED	ICE9B+	Branches mispredicted.
5'h10	FLOAT_B0	ICE9B+	Floating point instructions executed, bit 0. Note this includes all COP1 instructions, including load/stores.
5'h11	FLOAT_B1	ICE9B+	Floating point instructions executed, bit 1. Multiply by 2 and add bit 0 counter for total number of instructions. (Scaled in driver software, so read P1_FLOAT instead.)
5'h12	COP2_B0	ICE9B+	COP2 instructions executed, bit 0.

5'h13	COP2_B1	ICE9B+	COP2 instructions executed, bit 1.
5'h14	MSTALL	ICE9B+	Scheduling conflict M-stage stalls.
5'h15	DCEVICT	ICE9B+	D-Cache evicts.
5'h16	DCMISS	ICE9B+	D-Cache misses.
5'h17	TLBTRAP	ICE9B+	TLB traps.
5'h18	L2REQ	ICE9B+	Cachable L2 Cache requests
5'h19	L2MISS	ICE9B+	Cachable L2 Cache requests that miss in local L2
5'h1a	L2MISSALL	ICE9B+	Cachable L2 Cache requests that miss in all caches and fill from memory
5'h1b	FPARITH	ICE9B+	Floating point arithmetic instructions.
5'h1c	FPMADD	ICE9B+	Floating point multiply-add instructions.
5'h1d-5'h1f		ICE9B+	Reserved.

6.24.7 SCB Performance Events

The following events are trackable by SCB statistical event counting.

Enum

CpuScbEvent

Attributes

-descfunc

Constant	Mnemonic	Product	Definition
8'h00	CYCLES		Cpu cycles. Always counts.
8'h01	DCHIT		L1 D-Cache hits.
8'h02	DCMISS		L1 D-Cache misses.
8'h03	ICHIT		L1 I-Cache hits.
8'h04	ICMISS		L1 I-Cache misses.
8'h05	INSTNCOMPLETE		Instruction completed.
8'h06	ITLBHIT		Instruction TLB hits.
8'h07	ITLBMISS		Instruction TLB misses.
8'h08	DTLBHIT		Data TLB hits.
8'h09	DTLBMISS		Data TLB misses.
8'h0a	JTLBHIT		Joint TLB hits.
8'h0b	JTLBMISS		Joint TLB misses.
8'h0c	SLEEP		Sleep cycles. Cycles between WAIT instruction and interrupt or other wakeup.
8'h0d-8'h0f			Reserved.
8'h10			
8'h10	STALLR		R-stage pipeline stall.
8'h11	STALLR_DM		R-stage pipeline stall due to dispatch manager.
8'h12	STALLR_MD		R-stage pipeline stall due to multiply/divide.
8'h13	STALLR_CP		R-stage pipeline stall due to COP condition code.
8'h14	STALLR_DAT		R-stage pipeline stall due to data dependency. Includes data not ready, bypass not possible, and pending write-back stalls.
8'h17	STALLE		E-stage pipeline stall.
8'h18	STALLE_DCPRB		E-Stage DCache pipeline stall due to probe.
8'h19	STALLE_DCNPRB		E-Stage DCache pipeline stall due to non-probe. Sources include waiting for another fill, eviction buffer empty, read-after-write hazard prevention, etc.

8'h1a	STALLE_CP		E-stage pipeline stall due to COP condition code.
8'h1b	STALLE_CZ		E-stage pipeline stall due to coprocessor
8'h20	STALLM		M-stage pipeline stall.
8'h21	STALLM_CP		M-stage pipeline stall due to COP condition code.
8'h22	STALLM_DC		M-stage DCache pipeline stall.
8'h23	STALLM_LS		M-stage pipeline stall due to load-store.
8'h24	STALLM_MM		M-stage pipeline stall due to MMU.
8'h25	STALLM_CZ		M-stage pipeline stall due to coprocessor
8'h26	STALLM_DAT		M-stage pipeline stall due to data. Includes coprocessor data delivery conflicts, load/store data not ready, and WAW hazard delays.
8'h27-8'h2f			Reserved
8'h31	PROBE		Probes to L1.
8'h32	PROBE_HIT		Probes that hit L1.
8'h33	PROBE_DIRTY		Probes that hit dirty in L1.
8'h34	PROBE_LOCK		Probes that clear the lock bit.
8'h35	PROBE_WAIT		Cycles L2 is waiting for a probe to complete.
8'h38	LLHOLDOFF		Cycles the LL Timer is non-zero.
8'h40	RTN		Read return cycles.
8'h41	RTNL2_IO		Read return for I/O space. (Physical addr [35] set.)
8'h42	RTNL2_HIT		Read return came from local L2 cache.
8'h43	RTNL2_MISS		Read return did not come from local L2 cache.
8'h44	RTNL2_EXCL		Read return from local L2 exclusive.
8'h45	RTNL2_SHARED		Read return from local L2 shared.
8'h46	RTNL2_DIRTY		Read return from local L2 dirty.
8'h47	RTNL2_UPDATED		Read return from local L2 updated.
8'h48	RTNFR_COHO		Read return from coherence odd.
8'h49	RTNFR_COHE		Read return from coherence even.
8'h50	RTNFR_DMA		Read return from DMA.
8'h51	RTNFR_PCI		Read return from PCI.
8'h52	RTNFR_PS0		Read return from remote L2 0.
8'h53	RTNFR_PS1		Read return from remote L2 1.
8'h54	RTNFR_PS2		Read return from remote L2 2.
8'h55	RTNFR_PS3		Read return from remote L2 3.
8'h56	RTNFR_PS4		Read return from remote L2 4.
8'h57	RTNFR_PS5		Read return from remote L2 5.
8'h58	RTNFR_IO		Read return for IO transaction.
8'h60	RDQ1		Read queue entry 1 occupied.
8'h61	RDQ1S		Read queue shadow entry 1 occupied.
8'h62	RDQ2		Read queue entry 2 occupied.
8'h63	RDQ3		Read queue entry 3 occupied.
8'h64	WRQ2		Write queue entry 2 occupied.
8'h65	WRQ2S		Write queue shadow entry 2 occupied.
8'h66	WRQ3		Write queue entry 3 occupied.
8'h67	WRQ4		Write queue entry 4 occupied.
8'h70	INT0		Interrupt 0 cycles. Cycles cpu interrupt #0 asserted (not occurances), ignoring mask bit.
8'h71	INT1		Interrupt 1 cycles.
8'h72	INT2		Interrupt 2 cycles.
8'h73	INT3		Interrupt 3 cycles.
8'h74	INT4		Interrupt 4 cycles.
8'h75	INT5		Interrupt 5 cycles.

8'h76	INT6		Interrupt 6 cycles.
8'h77	INT7		Interrupt 7 cycles.
8'h78	INT		Interrupt cycles. Cycles asserted (not occurrences) across all types, ignoring mask bits.
8'hb0	IFETCHWT		Cycles of I-Stream Fetch Wait. Indicates the pipeline was empty, and data was not delivered by ICache. Generally indicates ICache miss delays or ITLB miss delays.
8'hb1	IFETCHWT8		A IFETCHWT of >= 8 cycles.
8'hb2	IFETCHWT16		A IFETCHWT of >= 16 cycles.
8'hb3	IFETCHWT24		A IFETCHWT of >= 24 cycles.
8'hb4	IFETCHWT32		A IFETCHWT of >= 32 cycles.
8'hb5	IFETCHWT48		A IFETCHWT of >= 48 cycles.
8'hb6	IFETCHWT64		A IFETCHWT of >= 64 cycles.
8'hb7	IFETCHWT96		A IFETCHWT of >= 96 cycles.
8'hb8	DATAWT		Cycles of Data Fetch Wait. Indicates a instruction was stalled waiting for source registers. Generally indicates DCache miss delays, DTLB miss delays, or other data dependant delays.
8'hb9	DATAWT8		A DATAWT of >= 8 cycles.
8'hba	DATAWT16		A DATAWT of >= 16 cycles.
8'hbb	DATAWT24		A DATAWT of >= 24 cycles.
8'hbc	DATAWT32		A DATAWT of >= 32 cycles.
8'hbd	DATAWT48		A DATAWT of >= 48 cycles.
8'hbe	DATAWT64		A DATAWT of >= 64 cycles.
8'hbf	DATAWT96		A DATAWT of >= 96 cycles.
			(Below events C0-DF correspond to the Cpu's internal performance counter 0 events, though with different numbering. They only count in user, supervisor, or kernel mode, as programmed based on the R_CpuPerfCount[0]/CP0 Reg25 register. The CPU internal counters may increment by 2 or 3 for some events. As the SCB can only increment by one, these events are split into _B0 and _B1 events. Count both then present to the user _B1*2+_B0, the result should be similar to the CPU internal count for the same event. In ICE9A for these SCB events to increment one of the Cpu's internal performance counters must be enabled. This restriction is removed in ICE9B.)
8'hc0(-8'hdf)	C0	ICE9B+	Core Perf 0 ENUM:CpuScbCoreEvent. See above note. See the CpuScbCoreEvent enumeration; it is inserted here to avoid duplication in this table.
8'hc0	P0_CYCLES	ICE9A	Perf 0 Cycles.
8'hc1	P0_INSFETCH_B0	ICE9A	Perf 0 Instructions fetched bit 0.
8'hc2	P0_INSFETCH_B1	ICE9A	Perf 0 Instructions fetched bit 1. Multiply by 2 and add bit 0 counter for total number of instructions.
8'hc3	P0_INSSCHED	ICE9A	Perf 0 Instructions scheduled.
8'hc4	P0_INSDUAL_B1	ICE9A	Perf 0 Dual issued instructions bit 1. Multiply by 2 and add bit 0 counter for total number of instructions. (Scaled in driver software, so read P0_INSDUAL instead.)
8'hc5	P0_INSEXEC_B0	ICE9A	Perf 0 Instructions executed bit 0.
8'hc6	P0_INSEXEC_B1	ICE9A	Perf 0 Instructions executed bit 1. Multiply by 2 and add bit 0 counter for total number of instructions. (Scaled in driver software, so read P0_INSEXEC instead.)
8'hc7	P0_LOAD	ICE9A	Perf 0 Load/pref/sync/cache ops.

8'hc8	P0_STORE	ICE9A	Perf 0 Stores.
8'hc9	P0_SC	ICE9A	Perf 0 Conditional stores.
8'hca	P0_SCFAIL	ICE9A	Perf 0 Conditional stores that fail.
8'hcb	P0_BRANCH	ICE9A	Perf 0 Branches executed.
8'hcc	P0_ICMISS	ICE9A	Perf 0 I-Cache misses.
8'hcd	P0_ITLBMISS	ICE9A	Perf 0 ITLB misses.
8'hce	P0_DTLBMISS	ICE9A	Perf 0 DTLB misses.
8'hcf-8'hdf		ICE9A	Reserved
			(Below events 30-3F correspond to the Cpu's internal performance counter 1 events, though with different numbering. They only count in user, supervisor, or kernel mode, as programmed based on the R_CpuPerfCount[1]/CP0 Reg25 register.)
8'he0(-8'hff)	C1	ICE9B+	Core Perf 1 ENUM:CpuScbCoreEvent. See above note. See the CpuScbCoreEvent enumeration; it is inserted here to avoid duplication in this table.
8'he0	P1_CYCLES	ICE9A	Perf 1 Cycles.
8'he1	P1_INSEXEC_B0	ICE9A	Perf 1 Instructions executed, bit 0.
8'he2	P1_INSEXEC_B1	ICE9A	Perf 1 Instructions executed, bit 1. Multiply by 2 and add bit 0 counter for total number of instructions. (Scaled in driver software, so read P1_INSEXEC instead.)
8'he3	P1_LOAD	ICE9A	Perf 1 Load/pref/sync/cache ops.
8'he4	P1_STORE	ICE9A	Perf 1 Stores.
8'he5	P1_SC	ICE9A	Perf 1 Conditional stores.
8'he6	P1_FLOAT_B0	ICE9A	Perf 1 Floating point instructions executed, bit 0.
8'he7	P1_FLOAT_B1	ICE9A	Perf 1 Floating point instructions executed, bit 1. Multiply by 2 and add bit 0 counter for total number of instructions. (Scaled in driver software, so read P1_FLOAT instead.)
8'he8	P1_COP2_B0	ICE9A	Perf 1 COP2 instructions executed, bit 0.
8'he9	P1_COP2_B1	ICE9A	Perf 1 COP2 instructions executed, bit 1.
8'hea	P1_MSTALL	ICE9A	Perf 1 Scheduling conflict M-stage stalls.
8'heb	P1_MISPRED	ICE9A	Perf 1 Branches mispredicted.
8'hec	P1_DCMISS	ICE9A	Perf 1 Data cache misses.
8'hed	P1_DCEVICT	ICE9A	Perf 1 Data cache line evicted.
8'hee	P1_TLBTRAP	ICE9A	Perf 1 TLB miss exception traps.
8'hef-8'hff		ICE9A	Reserved.

6.24.8 CpuConfig Register

Class

R_CpuConfig

Bit	Mnemonic	Access	Reset	Type	Definition
31	M	R	1		Indicates that the Config1 register is implemented.
30:28	K23	RW	X		Kseg2 and kseg3 cache coherency algorithm.
27:25	KU	RW	X		Useg/kuseg cache coherency algorithm.
24:22	LLTIME	RW	0		**Lock timer interval. 000=8 cycles, 001=16 cycles, in powers of 2 up to 111=1024 cycles. (SiCortex Change.)**
21	SB	R	X		SimpleBE bus mode is enabled.
20	ISD	RW	0		Instruction Scheduling Disable.
19	WC	RW	0		Unknown. Not documented, but implemented as RW bit in 5kf core.
17	DID	RW	0		Dual Issue Disable.
16	BM	R	X		Burst Mode.
15	BE	R	X		Big endian byte-ordering convention.
14:13	AT	R	2		Architecture Type.
12:10	AR	R	0		Architecture Revision.
9:7	MT	R	1		MMU Type.
2:0	K0	RW	2		Specifies the kseg0 cache coherency algorithm.

6.24.9 CpuConfig1 Register

Class

R_CpuConfig1

Bit	Mnemonic	Access	Reset	Type	Definition
30:25	MMUSizeM1	R	X		Number of entries in the TLB minus one.
24:22	IS	R	X		I-cache sets per way.
21:19	IL	R	X		I-cache line size.
18:16	IA	R	X		I-cache set associativity.
15:13	DS	R	X		D-cache sets per way.
12:10	DL	R	X		D-cache line size.
9:7	DA	R	X		D-cache set associativity.
6	C2	R	X		Coprocessor 2 implemented.
5	MD	R	X		MDMX ASE implemented.
4	PC	R	1		Performance Counter.
3	WR	R	1		Watch registers implemented.
2	CA	R	0		Code compression.
1	EP	R	1		EJTAG implemented.
0	FP	R	1		FPU implemented.

6.24.10 CpuConfig2 Register

Class

R_CpuConfig2

Bit	Mnemonic	Access	Reset	Product	Definition
31	M	R	1	twc9a+	Implemented.
30:28	TU	R	0	twc9a+	Tertiary cache control.
27:24	TS	R	0	twc9a+	Tertiary cache sets per way.
23:20	TL	R	0	twc9a+	Tertiary cache line size.
19:16	TA	R	0	twc9a+	Tertiary cache associativity.
15:12	SU	R	0	twc9a+	Secondary cache control.
11:8	SS	R	5	twc9a+	Secondary cache sets per way. Indicates 2K sets.
7:4	SL	R	5	twc9a+	Secondary cache line size. Indicates 64 byte lines.
3:0	SA	R	1	twc9a+	Secondary cache associativity. Indicates 2 associativities.

6.24.11 CpuFCCR Register

Class

R_CpuFCCR

Bit	Mnemonic	Access	Reset	Type	Definition
7:0	FCC	RW	X		Floating-point condition code.

6.24.12 CpuWatchLo Register

Class

R_CpuWatchLo

Bit	Mnemonic	Access	Reset	Type	Definition
63:3	VAddr	RW	X	uint64_t	Virtual Address.
2	I	RW	0		Watch exceptions are enabled for instruction fetches.
1	Rd	RW	0		Watch exceptions are enabled for loads.
0	Wr	RW	0		Watch exceptions are enabled for stores.

6.24.13 CpuWatchHi Register

Class

R_CpuWatchHi

Bit	Mnemonic	Access	Reset	Type	Definition
31	M	R	0		Only one pair of WatchHi/WatchLo registers are implemented.
30	G	RW	X		Global match.
23:16	ASID	RW	X		ASID.
11:3	Mask	RW	X		Bit mask that qualifies the address in the WatchLo register.

6.24.14 CpuFEXR Register

Class

R_CpuFEXR

Bit	Mnemonic	Access	Reset	Type	Definition
17:12	Cause	RW	X		Cause bits.
6:2	Flags	RW	X		Flag bits.

6.24.15 CpuXContext Register

Class

R_CpuXContext

Bit	Mnemonic	Access	Reset	Type	Definition
63:33	PTEBase	RW	X		Page Table Entry Base.
32:31	R	R	X		Region.
30:4	BadVPN2	R	X		BadVAddr register.

6.24.16 CpuDebug Register

Class

R_CpuDebug

Bit	Mnemonic	Access	Reset	Type	Definition
31	DBD	R	X		Debug Branch Delay.
30	DM	R	0		Debug Mode.
29	NoDCR	R	0		Dseg memory segment is present.
28	LSNM	RW	0		Load Store Normal Memory.
27	Doze	R	X		Processor was in low-power mode when a debug exception occurred.
26	Halt	R	X		Internal system bus clock was stopped when the debug exception occurred.
25	CountDM	R	1		Count Debug Mode.
23	MCheckP	RW	0		Machine Check Exception Pending.
22	CacheEP	RW	0		Cache Error Exception Pending.
21	DBusEP	RW	0		Data Bus Error Exception Pending.
18	DDBLImpr	R	X		Debug Data Break Imprecise.
17:15	EJTAGver	R	2		Version 2.
14:10	DExcCode	R	X		Debug Exception Code.
8	SSt	RW	0		Debug Single Step.
5	DINT	R	X		Debug Interrupt.
4	DIB	R	X		Debug Instruction Break.
3	DDBS	R	X		Debug Data Break Store.
2	DDBL	R	X		Debug Data Break Load.
1	DBp	R	X		Debug Breakpoint.
0	DSS	R	X		Debug Single Step.

6.24.17 CpuDEPC Register

Class

R_CpuDEPC

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	DEPC	RW	X	uint64_t	Debug Exception Program Counter.

6.24.18 CpuPerfCnt Register

Class

R_CpuPerfCnt

Bit	Mnemonic	Access	Reset	Product	Definition
31	M	R	1		Another pair of Performance Control and Counter registers implemented.
30	Wide	R	0	ICE9B+	Wide counters. Always 0 to indicate the counters are 32-bits wide. This bit is part of MIPS Release 2 architecture.
29:11					Reserved by architecture
10:5	Event6	RW	X	ICE9B+	Counter event enabled for this counter. See 6.24.5. Overlaps Event.
8:5	Event	RW	X	ICE9A	Counter event enabled for this counter. See 6.24.5. Overlaps Event6.
4	IE	RW	0		Counter interrupt enable. Because interrupts are level sensitive, clearing the enable near the time when the count will overflow may cause an interrupt that will disappear before the software services the interrupt. Generally software will ignore such interrupts.
3	U	RW	X		Count in User Mode.
2	S	RW	X		Count in Supervisor Mode.
1	K	RW	X		Count in Kernel Mode.
0	EXL	RW	X		Count when EXL.

6.24.19 CpuPerfVPC Register

Register 22, select 0 for event 0. Register 22, select 1 for event 1.

Class

R_CpuPerfVPC

Bit	Mnemonic	Access	Reset	Type	Definition
63:62	VPCH	R	X		**High bits of VPC.**
39:2	VPCL	R	X		**Event 0/1 Virtual Program Counter. For the last event 0/1 during SCB counting, the current virtual PCs.**
1:0					Reserved.

6.24.20 CpuPerfPEA Register

Register 22, select 2 for event 0. Register 22, select 3 for event 1.

Class

R_CpuPerfPEA

Bit	Mnemonic	Access	Reset	Type	Definition
63	L2HIT	R	X		**Last L2 hit. L2 cache indicated hit for the last L1 miss, during SCB counting.** Often wrong, see bug2674.
62:60	L2STATE	R	X		**Last L2 cache state. L2 cache state the last L1 miss came from, during SCB counting.** Often wrong, see bug2674.
59:56	L2STOP	R	X		**Last Bus stop. Bus stop number the last L1 miss was serviced by, during SCB counting. See CswStopNum.** Often wrong, see bug2674.
55:48	ASID	R	X		**Event 0/1 ASID. For the last event 0 during SCB counting, the ASID.**
47:36					Reserved.
35:5	PEA	R	X		**Event 0/1 Physical Effective Address. For the last event 0/1 during SCB counting, the current physical effective address of the last D-Cache hit or miss.** Note that this might not be the miss address, as a DC hit-under-miss following the miss will report the address of the DC hit.
4:0					Reserved.

6.24.21 CpuFENR Register

Class

R_CpuFENR

Bit	Mnemonic	Access	Reset	Type	Definition
11:7	Enables	RW	X		Enable bits.
2	FS	RW	X		Flush to Zero bit.
1:0	RM	RW	X		Rounding mode.

6.24.22 CpuErrCtl Register

Class

R_CpuErrCtl

Attributes

-kernel

Bit	Mnemonic	Access	Reset	Type	Definition
31	CorEna	RW	0		**Parity/ECC correction enable. SiCortex: Set to enable correction of ECC errors. Note ICE9A contains bug1965: reads of this bit are seen in bit 28.**
30	PO	R	0		**Parity Overwrite. SiCortex undefined behavior, must be zero.**
29	WST	RW	0		**Way Selection Test. SiCortex undefined behavior, must be zero.**
28	DetEna	RW	0		**Enable Parity/ECC reporting. (SiCortex addition) This bit is automatically cleared by HW before invoking the Cache error trap handler. The OS Cache Error Trap Handler needs to reenable reporting before returning. Note ICE9A contains bug1965: reads of this bit are seen in bit 31.**
27	DriveBadDat1	RW	0		**Flip bit 1 in all ECC generation trees, for diagnostic ECC error generation. (SiCortex addition)**
26	DriveBadDat0	RW	0		**Flip bit 0 in all ECC generation trees, for diagnostic ECC error generation.(SiCortex addition)**
25:8					Reserved.
7:0	P	R	0		**Parity bits read or written to a cache data RAM. SiCortex undefined behavior, must be zero.**

6.24.23 CpuCacheErr Register

Class

R_CpuCacheErr

Bit	Mnemonic	Access	Reset	Type	Definition
31	ER	R	X		Error Reference.
29	ED	R	X		Error Data. (Single or double)
28	ET	R	X		Error Tag. (Single or double)
25	EB	R	X		Additional data cache error.
24	EF	R	X		Error Fatal. **SiCortex: Only set for double bit errors.**
22	EW	R	X		Error Way. **SiCortex: Often incorrect for D-Cache, bug1575.**
21:20	Way	R	X		Way.
15:0	Index	R	X		Index. **SiCortex: Often incorrect for D-Cache probes, bug1575.**

6.24.24 CpuTagLo Register

Class

R_CpuTagLo

Bit	Mnemonic	Access	Reset	Type	Definition
31:8	PTagLo	RW	X		Specifies the upper address bits for the cache tag.
7:6	PState	RW	X		Valid dirty line.
5	L	RW	X		State of the lock bit for the cache line.
0	P	RW	X		Parity bit for the cache tag.

6.24.25 CpuDataLo Register

Class

R_CpuDataLo

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	Data	RW	X	uint64_t	Data read from the data array of the cache.

6.24.26 CpuDataHi Register

Class

R_CpuDataHi

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Data	RW	X		High-order data read from the cache data array.

6.24.27 CpuErrorEPC Register

Class

R_CpuErrorEPC

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	ErrorEPC	RW	X	uint64_t	Error Exception Program Counter.

6.24.28 CpuDESAVE Register

Class

R_CpuDESAVE

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	DESAVE	RW	X	uint64_t	Simple Read/Write register.

6.24.29 CpuDCR Register

Class

R_CpuDCR

Bit	Mnemonic	Access	Reset	Type	Definition
29	ENM	R	X		Endianess in which the processor is running in Kernel and Debug Modes.
17	DataBrk	R	X		Data hardware breakpoint is implemented.
16	InstBrk	R	X		Instruction hardware breakpoint is implemented.
4	IntE	RW	1		Hardware and software interrupt enable for Non-Debug Mode.
3	NMIE	RW	1		Non-Maskable Interrupt (NMI) enabled for Non-Debug Mode.
2	NMIpend	R	0		Indicates pending NMI.
1	SRstE	RW	1		Soft reset is fully enabled.
0	ProbEn	R	X		Probe services accesses to dmseg Reads as zero.

6.24.30 CpuFCSR Register

Class

R_CpuFCSR

Bit	Mnemonic	Access	Reset	Type	Definition
31:25,23	FCC	RW	X		Floating-point condition codes.
24	FS	RW	X		Flush to Zero.
22	FO	RW	X		Flush Override.
21	FN	RW	X		Flush to Nearest.
17:12	Cause	RW	X		Cause bits.
11:7	Enables	RW	X		Enable bits.
6:2	Flags	RW	X		Flag bits.
1:0	RM	RW	X		Rounding mode.

6.24.31 CpuIBS Register

Class

R_CpuIBS

Bit	Mnemonic	Access	Reset	Type	Definition
30	ASIDsup	R	1		ASID compare is supported in instruction breakpoints.
27:24	BCN	R	4		Number of instruction breakpoints implemented.
3:0	BS30	RW	FW0		Break status.
3	BS3	RW	FW0		Break status. Overlaps BS30.
2	BS2	RW	FW0		Break status. Overlaps BS30.
1	BS1	RW	FW0		Break status. Overlaps BS30.
0	BS0	RW	FW0		Break status. Overlaps BS30.

6.24.32 CpuIBA Register

Class

R_CpuIBA

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	IBA	RW	X	uint64_t	Instruction breakpoint address for condition.

6.24.33 CpuIBM Register

Class

R_CpuIBM

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	IBM	RW	X	uint64_t	R/W Instruction breakpoint address mask for condition.

6.24.34 CpuIBASID Register

Class

R_CpuIBASID

Bit	Mnemonic	Access	Reset	Type	Definition
7:0	ASID	RW	X		Instruction breakpoint ASID value for compare.

6.24.35 CpuIBC Register

Class

R_CpuIBC

Bit	Mnemonic	Access	Reset	Type	Definition
23	ASIDuse	RW	X		Use ASID value in compare for instruction breakpoint.
2	TE	RW	0		Use instruction breakpoint n as triggerpoint.
0	BE	RW	0		Use instruction breakpoint n as breakpoint.

6.24.36 CpuDBS Register

Class

R_CpuDBS

Bit	Mnemonic	Access	Reset	Type	Definition
30	ASIDsup	R	1		ASID compare is supported in data breakpoints.
29	NoSVmatch	R	0		Value compare on a store is supported in data breakpoints.
28	NoLVmatch	R	0		Value compare on a load is supported in data breakpoints.
27:24	BCN	R	2		Number of data breakpoints implemented.
1:0	BS10	RW	X		Number of BS bits implemented corresponds to the number of breakpoints indicated.

6.24.37 CpuDBA Register

Class

R_CpuDBA

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	DBA	RW	X	uint64_t	Data breakpoint address for condition.

6.24.38 CpuDBM Register

Class

R_CpuDBM

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	DBM	RW	X	uint64_t	Data breakpoint comparison mask.

6.24.39 CpuDBASEID Register

Class

R_CpuDBASEID

Bit	Mnemonic	Access	Reset	Type	Definition
7:0	ASID	RW	X		Data breakpoint ASID value for compare.

6.24.40 CpuDBC Register

Class

R_CpuDBC

Bit	Mnemonic	Access	Reset	Type	Definition
23	ASIDuse	RW	X		Use ASID value in compare.
21:14	BAI70	RW	X		Byte access ignore.
13	NoSB	RW	X		Condition can be fulfilled on store access.
12	NoLB	RW	X		Condition can be fulfilled on load access.
11:4	BLM70	RW	X		Compare corresponding byte lane.
2	TE	RW	0		Use data breakpoint as triggerpoint.
0	BE	RW	0		Use data breakpoint as breakpoint.

6.24.41 CpuDBV Register

Class

R_CpuDBV

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	DBV	RW	X	uint64_t	Data breakpoint data value for condition.

6.24.42 CpuIndex Register

Class

R_CpuIndex

Bit	Mnemonic	Access	Reset	Type	Definition
31	P	R	X		Probe Failure.
5:0	Index	RW	X		Index to the TLN entry used by the TLB read.

6.24.43 CpuRandom Register

Class

R_CpuRandom

Bit	Mnemonic	Access	Reset	Type	Definition
5:0	Random	R	X		TLB Random Index.

6.24.44 CpuEntryLo Register

Class

R_CpuEntryLo

Bit	Mnemonic	Access	Reset	Type	Definition
29:6	PFN	RW	X		Page Frame Number.
5:3	C	RW	X		Coherency attribute of the page.
2	D	RW	X		Dirty bit.
1	V	RW	X		Valid bit.
0	G	RW	X		Global bit.

6.24.45 CpuContext Register

Class

R_CpuContext

Bit	Mnemonic	Access	Reset	Type	Definition
63:23	PTEBase	RW	X		OS Use.
22:4	BadVPN2	RW	X		Virtual address updated on exceptions.

6.24.46 CpuPageMask Register

Class

R_CpuPageMask

Bit	Mnemonic	Access	Reset	Type	Definition
24:13	Mask	RW	X		Mask indicating which bits of VA must match.

6.24.47 CpuWired Register

Class

R_CpuWired

Bit	Mnemonic	Access	Reset	Type	Definition
5:0	Wired	RW	0		TLB wired boundary.

6.24.48 CpuBadVAddr Register

Class

R_CpuBadVAddr

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	BadVAddr	R	X	uint64_t	Virtual address that caused an exception.

6.24.49 CpuFIR Register

Class

R_CpuFIR

Bit	Mnemonic	Access	Reset	Type	Definition
19	Cpu3D	R	0		MIPS-3D ASE is implemented.
18	PS	R	0		Paired-single floating-point implemented.
17	D	R	1		Double-precision floating-point implemented.
16	S	R	1		Single-precision floating-point implemented.
15:8	ProcessorID	R	0x81		Floating-point processor type.
7:0	Revision	R	X		Matches CP0 PRId register.

6.24.50 CpuCount Register

Class

R_CpuCount

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Count	RW	X		Interval counter. Counts every other pclk. **Zeroed, then starts counting after reset. This allows all CPUs to have the same zero start time for exact cycle release-from-barrier.**

6.24.51 CpuEntryHi Register

Class

R_CpuEntryHi

Bit	Mnemonic	Access	Reset	Type	Definition
63:62	R	RW	X		Virtual memory region, corresponding to VA63:62.
61:40	Fill	R	0		Fill bits.
39:13	VPN2	RW	X		VA39:13 of the virtual address.
7:0	ASID	RW	X		Address Space Identifier.

6.24.52 CpuCompare Register

Class

R_CpuCompare

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Compare	RW	X		Interval count compare value.

6.24.53 CpuStatus Register

Class

R_CpuStatus

Bit	Mnemonic	Access	Reset	Type	Definition
31	CU3	RW	X		Coprocessor Usable.
30	CU2	RW	X		Coprocessor Usable.
29	CU1	RW	X		Coprocessor Usable.
28	CU0	RW	X		Coprocessor Usable.
27	RP	RW	X		Reduced power.
26	FR	RW	X		Floating-point register mode.
25	RE	RW	X		Reverse Endian.
24	MX	RW	X		Enable access to MDMX resources on processors.
23	PX	RW	X		Processor Extension.
22	BEV	RW	X		Bootstrap Exception Vector.
21	TS	RW	X		TLB Shutdown.
20	SR	RW	X		Soft Reset.
19	NMI	RW	X		Non-maskable Interrupt.
15:8	IM	RW	X		Interrupt Mask.
7	KX	RW	X		Kernel Extension.
6	SX	RW	X		Supervisor Extension.
5	UX	RW	X		User Extension.
4:3	KSU	RW	X		Base mode.
2	ERL	RW	X		Error Level.
1	EXL	RW	X		Exception Level.
0	IE	RW	X		Interrupt Enable.

6.24.54 CpuCause Register

Class

R_CpuCause

Bit	Mnemonic	Access	Reset	Type	Definition
31	BD	R	X		Branch Delay.
29:28	CE	R	X		Coprocessor Exception.
23	IV	RW	X		Interrupt Vector.
22	WP	RW	X		Watch Postponed.
15	IP7	R	X		Interrupt Pending.
14	IP6	R	X		Interrupt Pending.
13	IP5	R	X		Interrupt Pending.
12	IP4	R	X		Interrupt Pending.
11	IP3	R	X		Interrupt Pending.
10	IP2	R	X		Interrupt Pending.
9	IP1	RW	X		Interrupt Pending.
8	IP0	RW	X		Interrupt Pending.
6:2	ExcCode	R	X		Exception Code.

6.24.55 CpuEPC Register

Class

R_CpuEPC

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	EPC	RW	X	uint64_t	Exception Program Counter.

6.24.56 CpuPRId Register

Class

R_CpuPRId

Attributes

-kernel

Bit	**Mnemonic**	**Access**	**Reset**	**Type**	**Definition**
31:24	CompanyOptions	R	X		Available to the CPU core user for company-dependent options. Overlaps Allowed.
31	OneCpu	R	X		**Single core mode. Set in simulation model only.**
28:24	CoreNum	R	X		**Core number (0-5) on the chip. (SiCortex enhancement.)**
23:16	CompanyID	R	14		**Company that designed or manufactured processor. 1=MIPS, 14=SiCortex. (SiCortex change.)**
15:8	ProcessorID	R	pins	AddrProduct	**Type of processor. Returns ICE9, ICE9B, etc. (SiCortex change.)**
7:0	Revision	R	1		Revisions of the same processor type.

Note: Revision not incremented between ICE9A and ICE9A1. To determine ICE9A vs ICE9A1 read Rev field of SCB register R_ScbChipRev.

6.24.57 Ecc Injection Magic Register

The cache ECC Magic registers are used to generate L1 ECC errors. This is implemented only in the verification model, for testing purposes.

Register

R_CpuxEccInjMagic

Attributes

-noregtest -noregdump

Address

0x00_0400 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31	Go	W	0		When written one, toggle bit as specified.
30	Icache	W	0		Write ICache, else if zero write DCache.
29	Tag	W	0		Write Tag RAM, else if zero write data RAM.
28:26					Reserved
25:24	Way	W	0		Cache way to write.
23:16	Bitnum	W	0		Bit number in physical RAM to toggle. Includes both data, ecc, and parity bits, where enumberation depends on internal RAM organization.
15:14					Reserved
13:0	Index	W	0		Cache index to write.

6.25 EJTAG Registers and Definitions

6.25.1 EJTAG TAP Instructions

Enum

CpuTapInstr

Constant	**Mnemonic**	**Definition**
5'h1	IDCODE	Selects Device id
5'h3	IMPCODE	Selects Implementation register
5'h8	ADDRESS	Selects Address register

5'h9	DATA	Selects Data register
5'hA	CONTROL	Selects EJTAG Control register
5'hB	ALL	Selects Address, Data and EJTAG Control
5'hC	EJTAGBOOT	Enables debug exception after reset
5'hD	NORMALBOOT	Disables debug exception after reset
5'hE	FASTDATA	Selects the Data and Fastdata register
5'h1F	BYPASS	High-order portion of the TLB entry

6.25.2 CpuTapIDCODE Register

Class

R_CpuTapIDCODE

Attributes

-tapSize=32

Bit	Mnemonic	Access	Reset	Type	Definition
31:28	Version	R	X		Identifies the version of a specific device. In ICE9A turns one. In ICE9B and followons returns processor nu ber.
27:12	Part	R	X	**AddrProduct**	**Identifies the part number of a specific vice. In ICE9A contains value of ICE9_CPU ICE9_CPU5 as appropriate. Later passes cont ICE9*_CPU.**
11:1	ManufID	R	**SICORTEX**	**AddrTapMfgr**	**Identifies the manufacturer identity code of a sp cific device,.**

6.25.3 CpuTapIMPCODE Register

Class

R_CpuTapIMPCODE

Attributes

-tapSize=32

Bit	Mnemonic	Access	Reset	Type	Definition
31:29	EJTAGver	R	X		EJTAG version implemented.
24	DINTsup	R	X		Support for the DINT signal from the probe.
22:21	ASIDsize	R	2		Size of the ASID field.
16	MIPS16	R	0		MIPS16 ASE is supported.
14	NoDMA	R	1		Indicates no EJTAG DMA support.
0	MIPS64	R	1		64-bit processor.

6.25.4 CpuTapDATA Register

Class

R_CpuTapDATA

Attributes

-tapSize=64

Bit	Mnemonic	Access	Reset	Type	Definition
63:0	Data	RW	X	uint64_t	Data used by processor access.

6.25.5 CpuTapADDRESS Register

Class

R_CpuTapADDRESS

Attributes

-tapSize=36

Bit	Mnemonic	Access	Reset	Type	Definition
35:0	Address	RW	X	uint64_t	Address used by processor access.

6.25.6 CpuTapECR Register

Class

R_CpuTapECR

Attributes

-tapSize=32

Bit	Mnemonic	Access	Reset	Type	Definition
31	Rocc	RW	1		Soft reset has occurred since last bit cleared.
30:29	Psz	RW	X		Size of pending access. 0=byte, 1=HW, 2=W, 3=DW
22	Doze	R	0		Processor in low-power mode.
21	Halt	R	1		Internal clock is running.
20	PerRst	RW	0		Peripheral reset.
19	PRnW	R	X		Read not write processor access.
18	PrAcc	RW	0		Pending processor access.
16	PrRst	RW	0		Apply processor reset.
15	ProbEna	RW	X		Probes will be serviced by EJTAG.
14	ProbTrap	RW	X		Relocates debug exception vector.
12	EjtagBrk	RW	X		Requests debug exception.
3	DM	R	0		In debug mode.

6.25.7 CpuTapFASTDATA Register

Class

R_CpuTapFASTDATA

Attributes

-tapSize=1

Bit	Mnemonic	Access	Reset	Type	Definition
0	SPrAcc	RW	X		Zero if processor action completed. (See documentation.)

6.26 Cpu Implementation-Only Definitions

6.26.1 Request Commands

These encodings are used for cpu_cac_reqCmd_pr.

Enum

CpuReqCmd

Constant	Mnemonic	Product	Definition
3'b000		TWC9A+	Reserved. (If we remove Valid, this becomes the idle)

3'b010	WR	TWC9A+	Write.
3'b011	INV	TWC9A+	Invalidated. (Reserved for later)
3'b100	READ	TWC9A+	Read.
3'b101	PREF	TWC9A+	Prefetch.
(else)		TWC9A+	Reserved.

6.27 Cac Registers and Definitions

6.27.1 Probe Queue Handler States

This is the encoding for the probe queue handler state machine in the CAC portion of the processor segment.

Enum

CacPrbQState

Constant	Mnemonic	Definition
5'h0	POLL	Look for next entry on the queue
5'h1a	POLL2	Wait for the L1/L2 pipeline to drain
5'h1	INV_WT	We found a PRBINV, wait to do L2 lookup
5'h2	INV_LU	Perform lookup/inval in L2
5'h19	INV_LU2	Check tag compare result from L2
5'h3	BRD_WT	We found a PRBBRD, wait to do L2 lookup
5'h4	BRD_LU	Perform lookup in L2
5'h14	BRD_LU2	Check tag compare result from L2
5'h10	BRD_L2WB	Wait for L2 to dump into writeback buffer
5'h5	BRD_WBWT	Wait for writeback to complete
5'h6	BWT_WT	We found a PRBBWT, wait to do L2 lookup
5'h7	BWT_LU	Perform lookup in L2
5'h17	BWT_LU2	Check tag compare result from L2
5'h18	BWT_PWT	Wait for L1 probe invalidate to complete
5'h8	BWT_GO	Tell originator to launch fill
5'h9	BWT_FWT	Wait for fill to complete
5'ha	WIN_WT	We found a PRBWIN, wait to do L2 lookup
5'hb	WIN_LU	Perform lookup in L2
5'h1b	WIN_LU2	Check tag compare result from L2
5'h12	WIN_L2WB	Wait for L2 to dump into writeback buffer
5'hc	WIN_WBWT	Wait for writeback to complete
5'hd	SHR_WT	We found a PRBSHR, wait to do L2 lookup
5'he	SHR_LU	Perform lookup in L2
5'h1e	SHR_LU2	Check tag compare result from L2
5'h11	SHR_L2WB	Wait for L2 to dump into writeback buffer
5'hf	SHR_WRSTWT	Wait for writeback to complete
5'h1c	L1PRBDN_WT	Wait for L1 probes to complete.
5'h13	CHK_NHACK	Wait for NOHIT to complete in CMX

6.27.2 Processor Interface Ready State Machine

Enum

CacRdyState

Constant	Mnemonic	Definition
4'h0	IDLE	Wait for the next request or a pause

4'h1	BP1	We're handling a block transfer, first tic done
4'h2	BP2	We're handling a block transfer, second tic done
4'h3	BP3	We're handling a block transfer, third tic done
4'h4	SP1	One tic of pause after a block transfer
4'h5	EP	End of Pause interval, look for next thing to do
4'h6	PAUSED	Pausing to honor BIU pause request from CTL or DAT unit
4'h7	PREPAUSE	We're about to pause, but we should check first to allow one last read to sneak in, if necessary.
4'hE	PREDOP1	We'd like to send out a pending op, but we need to wait two tics.
4'hF	PREDOP2	We'd like to send out a pending op, but we need to wait one more tic.
4'h8	DOPEND1	Pausing to complete pending read operations
4'h9	DOPEND2	It takes two tics to send out a pending read operation
4'hA	IP1	Pausing after an IO access
4'hB	IP2	Still pausing after an IO access
4'hC	IP3	Still pausing after an IO access or pending read op
4'hD	IP4	Still pausing after an IO access or pending read op

6.27.3 L2 Cache Pause During Fill State Machine

Enum

CacDpseState

Constant	Mnemonic	Definition
2'h0	IDLE	Wait for a new data block to arrive
2'h1	WT0	Wait for either BIUPaused or the last stage of FillIdx
2'h2	WT4PSED	Wait for BIUPaused to be asserted
2'h3	WT4FIDX	Wait for the last stage of FillIdx

Chapter 7

L2 Cache Coherence and Switch

by Jud Leonard and Matt Reilly.
[$Id: L2Cache.lyx 49898 2008-01-22 14:26:37Z zeno $]

7.1 Summary

The ICE9 node chip implements a 1.5 MByte L2 mixed instruction and data cache that is accessible from all six CPU cores, PCI-Express, and the DMA engine. The L2 cache is split into six segments, each closely connected with a single processor. Each L2 cache segment is 2-way set associative with a 64 byte line size, with writeback policy and allocation on read or write miss. It acts as a proper superset of the L1 data caches in the cores, and maintains coherence among them by enforcing exclusive ownership of writable blocks. The L2 supports coherent shared access among the cores without reference to main memory.

This section describes the Central Cache Switch (CSW) and the protocol that manages cache coherence and data movement among the processors and I/O devices on the ICE9 node. The first sections of this chapter give a general outline of the approach and present some notes on how we got here. The latter sections (beginning with Section 7.10) present detailed descriptions of transaction flows and responses.

For a more detailed outline of the Processor to L2 organization, see Chapter 6. For an explanation of the DMA interface to the L2 and CSW, see Chapter 5. For more information on the PCI Express controller and other I/O devices, see Chapters 15,13, 14, and 10.

7.2 Differences, Bugs, and Enhancements

7.2.1 Product and Chip Pass Differences

1. TWC9A's L2 cache is part of the new IceT core, and is described in a different document.
2. TWC9A adds the CswStopNumTwc and CswTidTwc enumeration to support more cores, and more TIDs per core, bug3377.
3. NEED IMPL: TWC9A fixes the R_CacxIntCr[#]_Overflow bit being mis-cleared when clearing R_CacxIntCr[#]_Active, bug3165.
4. NEED IMPL: The R_CohxEccMode_CorEna bit must be set whenever the ICE9 caches are active, bug1990.
5. NEED IMPL: TWC9A pushes IO writes instead of using a special command, bug4898.
6. NEED IMPL: TWC9A removes SPCL in favor of IO writes, bug4899.
7. NEED IMPL: TWC9A stalls issuing probes to avoid large per-cpu probe queues.

7.2.2 Known Bugs and Possible Enhancements

7.3 L2 Cache Features

The L2 cache stores 1.5 MB of data. It is structured as six 256 KB cache segments to provide sufficient bandwidth for 6 cores, and to minimize the typical access latency. Each segment is 2-way set associative. The cache is interfaced to two DDR2 SDRAM memory controllers, interleaved on the cache line size, 64 bytes.

- Line size = 64 Bytes (2^6) plus ECC on 8-byte doublewords
- Number of tags = 24K ($3 \star 2^{13}$) total
- Associativity = 6 Segments, 2 way associative.
- Index size = 11 bits {(address <26:17> xor <16:7>), address <6>}
- Tag, state, and all data are ECC protected
- Replacement = LRU nearest requestor
- Physical Address = 36 bits
- Protocols = Snooping, Writeback, Subset

Every processor request is attempted first in the local L2 segment. If it misses, the request is directed to one of the coherence controllers (at the memory interface), as selected by bit 6 of the address. The request must arbitrate for use of the memory request/address bus toward the selected controller. The coherence controller looks for the requested address in a duplicate tag store (the master); it may match in one or more of the tags corresponding to other processors. In the event of a hit, the controller redirects the request to the hit segment, which will return the block to the requestor and, in the case of a data-stream fetch, transfer ownership to the recipient.

7.3.1 Terminology

Block The unit of memory identified by one tag in the L1 cache, consisting of 4 doublewords (32 bytes) with byte parity. Synonymous with half-line.

Clean The state of a memory block which is known to be unchanged with respect to the value in memory. A clean block can safely be discarded.

Dirty The state of a memory block which has been modified since it was read from memory. It must be written back to memory (*victimized*) before its space in the cache is reclaimed. Synonymous with *Modified*.

Doubleword 8 bytes (64 bits). The standard size of data values in the 5Kf microprocessor, and the width of most data busses in the chip.

Exclusive The state of a cache block which ensures that it belongs to exactly one L2 segment and possibly the associated L1. The processor is permitted to modify a block if and only if it is in the exclusive state. It is allowed that a block be in only one segment without exclusive state, but not allowed to have exclusive state when there is a copy in more than one segment.

Line The unit of memory identified by one tag in the L2 cache. It consists of 8 doublewords (64 bytes) with ECC on each doubleword; equal to two blocks.

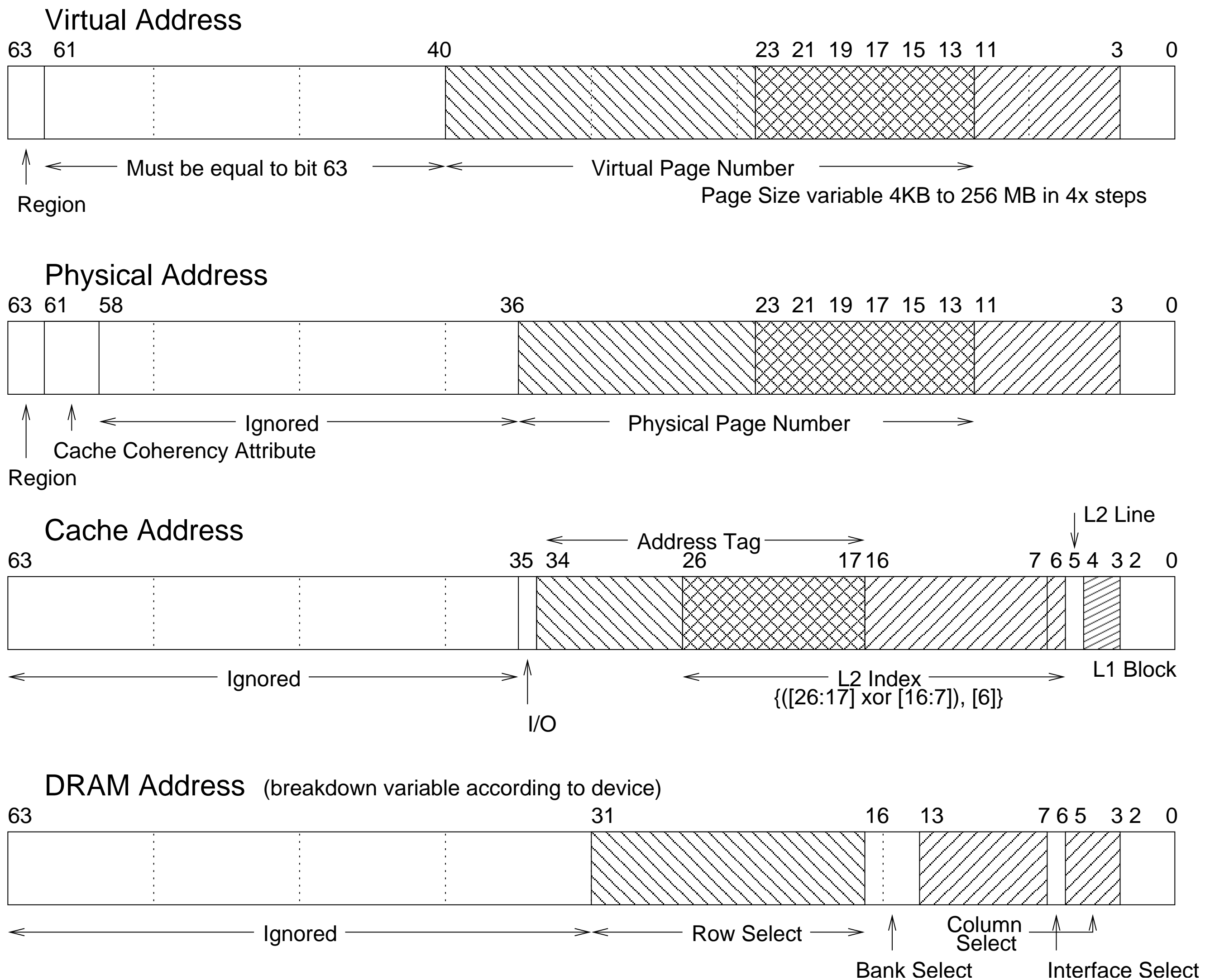
Segment One of the six 256 KB partitions of the L2 cache, consisting of a 2-way set associative cache with 64-byte lines and 2K sets. Each segment stores lines that have been accessed by the processor with which it is paired; data in any segment can be used to satisfy a cache miss, and writes are kept coherent among segments.

Shared The complement of exclusive state; a block in shared state is readable to any processor's instruction cache, but must be transitioned to exclusive state before it can be accessed by the data cache (and therefore written). It is possible for a block to be in shared state while being in only one segment.

Tag The auxilliary information stored with each line of a cache, indicating where that line belongs in main memory and its state with respect to memory.

Updated The state of a cache block after it has been written by the currently owning processor. That is, a block enters into an L2 segment in the EXCLUSIVE or DIRTY state. If the block is then written by the associated processor, it enters the DIRTY and UPDATED state. (Updated or Dirty blocks must be written to memory when they are evicted.) The Updated state is left over from an earlier complex scheme for maintenance of the LoadLinked/StoreConditional state. See Sections 7.8.1 and 6.6.10.

Figure 7.1: Address Partitions



7.3.2 Unusual Features

For those familiar with other cache designs, this one holds few surprises. It can be understood as six processors with separate snooping L2 caches. The major difference is that snooping uses a central "coherence controller" which keeps the master tags and victim buffer. The coherence controller maintains an accurate representation of the contents of all the cache segments, and need not take cycles from the segments unless a state change is required.

It is also unusual that this design does not support the "shared" state for data blocks (it does allow shared instruction blocks). The drive behind this decision comes from the fact that the MIPS L1 design does not provide a shared state separate from exclusive: the data cache will permit a write to any block it holds. We thought about redesigning the dcache controller, but at this point it doesn't appear that the performance impact of shuttling blocks between segments is so severe as to justify the risk and design effort.

7.3.3 Error Control

The L2 cache data and tag arrays are protected by a single-error-correcting, double-error-detecting (SEC/DED) Error Correcting Code which requires 8 ECC bits for each 64-bit doubleword of data. The normal read access path allows time for detecting and correcting errors in the tag or data arrays.

The cores expect parity on data blocks. L2 reads will correct and report single-bit errors, and present the corrected doubleword with valid parity. L2 writes will check parity as presented by the processor, and compute ECC.

7.4 Processor to L2 Cache Interface

NOTE: This section is dated. See the processor chapter 6for the current interface description.

7.5 Major Blocks and the General Approach

The L2/CSW implements a split transaction MESI (Modified, Exclusive, Shared, Invalid) cache coherence protocol. Each node on the daisy-chained pair of buses is connected at a “bus stop” and may initiate requests via the chain to any other node. Memory acceses are all sequenced through one of two coherence controllers. (Each controller is responsible for one of the two DIMM slots.) A fill request (caused by an L2 miss) is sent to the appropriate coherence controller and checked against its shadow copy of each processor’s L2 tag array. If the required block is not found in any other processor’s segment, the request is satisfied by the associated DRAM controller.

If the coherence widget finds a tag match in some processor’s L2 segment, the request will be forwarded to the appropriate processor and ownership will be transfered, if necessary.

In addition to normal cache transactions, the CSW and L2 protocols support block read and write operations from I/O and fabric devices. That is, the DMA engine – for example – may write an entire 64 byte block to physical memory. If the block is currently cached by a processor segment, the DMA engine will transfer its data directly to the L2 cache.

The following sections introduce the basic components and operations in the L2 CSW and Coherence widgets. More detailed information is presented in Section 7.10.

7.5.1 Supported Operations

Each processor may originate memory read and write transactions. Each memory transaction moves 64 bytes to and from a DRAM unit or another processor. A processor may have no more than 1 such transaction outstanding. L2 cache fills that may require victimization of a block will cause a processor segment to initiate a read-with-victimization operation (RDV or RDSV). Such operations count as one transaction, though the processor segment will write one block to memory and receive a second block for the fill.

The DMA engine and the PCI express controller may initiate block transfers of 32 or 64 bytes. Block read operations transfer data from DRAM if it is not cached, or are forwarded to the appropriate L2 cache segment. Block read transfers cause no change in ownership of the block – it stays in the owner’s cache. Block read operations are always 64 bytes long. Block write transfers may either send data to the DRAM or – if the block is cached – will overwrite the cached copy. Again, block write transfers cause no change in ownership of the block, and are atomic as far as a processor may observe.

Any unit on the CSW may originate and may accept I/O read and write transactions. All I/O transfers are 8 bytes long.

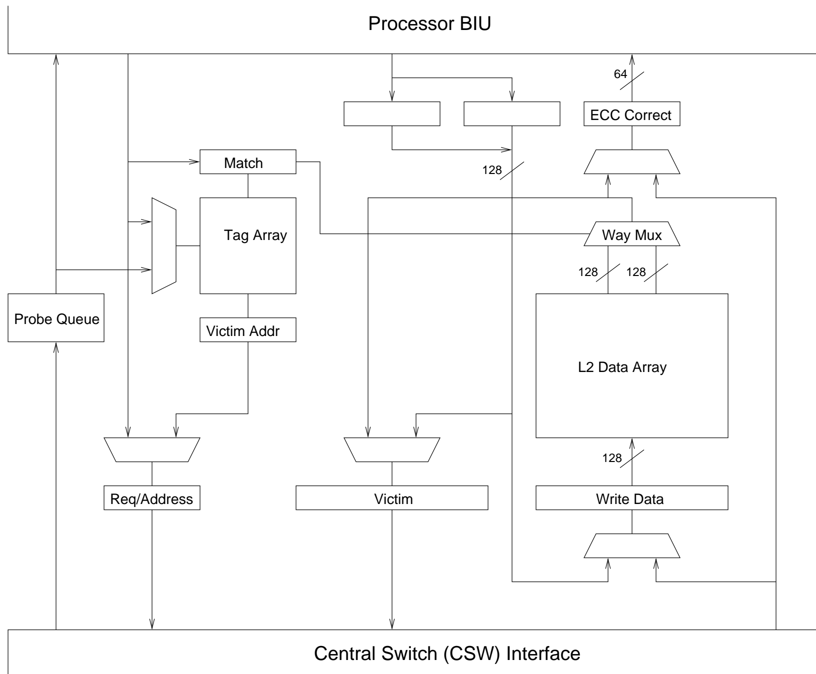
Processor segments must accept interrupt delivery transactions from any other unit on the CSW.

Any unit may accept special accelerated I/O write transfers, via the SPCL transaction. However, only the DMA engine supports SPCL, so SPCL to any other device is unsupported. (See Section 7.10.6.)

7.5.2 Per-Processor Segment

Figure 7.2 shows the structure of one segment of the L2 cache, standing between the Processor’s Bus Interface Unit and the Central Switch which connects all segments to the Coherence Controllers, and through them, to the memories.

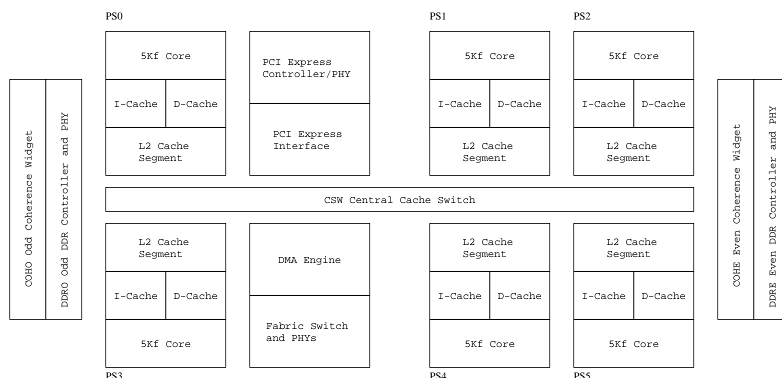
Figure 7.2: Segment Block Diagram (See Chapter 6.)



7.5.3 Bidirectional spine structure

Each processor communicates with memory and I/O through its associated L2 segment. The L2 caches, the DMA engine, and PCI-express interfaces share two busses, one to each of the coherence controllers. Processors use a 64-bit interface at 500 MHz, which is converted at the interface to 128 bits at 250 MHz in the L2 segment and on the Even and Odd-bound busses. (We don't use the more obvious East and West directions for historical reasons. The Even bound bus chain carries data from each bus stop (connection point) to the Even bank of memory (address[6] = 0) on the east side of the die. The Odd bound bus carries data from each bus stop to the Odd bank (address[6] = 1) on the west side of the die.)

Figure 7.3: Chip Floorplan



A very rough floorplan is shown in Figure 7.3. The arrangement and order of units along the CSW may change as we refine the routing.

The floorplan arrangement is chosen to put the major pin fields along edges of the chip: DDR memory interfaces (100+ pins each) on east and west sides, PCI-Express (~32 pins) on the north, and DMA Engine/Switch (~120 pins) on the south. The data arrays are arranged to group in each array bits which will be read and written simultaneously, and to line up arrays so that common address and data wires are straight. CSW busses extend the width of the chip, to reach all RAM arrays they must touch, and to be accessible to the processor input/output ports arrayed horizontally across the die. The CSW buses provide the principle medium for memory sharing among

processors.

7.5.4 Tags

Each line in the L2 cache is associated with a tag, which includes the high-order physical address bits identifying the cached memory block, plus dirty state and ECC bits. Each tag is stored twice: once (the "local" copy) in the cache segment close to the processor it primarily serves, and once (the "master") in one of the coherence controllers associated with each memory interface (selected by address bit 6). The local segment also keeps track of the most recently used way of each set, for use in replacement decisions.

The master tags are consulted when any reference misses in the local segment; if they show that the referenced block exists in another cache, the block is obtained from there rather than memory. A block may be exclusive (and therefore writable) in one segment, or shared (and therefore read-only) in several segments. To exclude the rare possibility that a line is dirty in several segments, we will victimize any dirty block when it is read for the i-cache.

7.5.5 Hashed Index

The L2 cache and tag arrays are addressed by physical address bits 16:7 XOR 26:17 catenated with bit 6; the tag arrays store bits 34:17. Victim addresses are reconstructed by using bits 34:17 from the tag, and XOR'ing the array index with bits 26:17 of the tag. Bit 6 is excluded from the tag hash as we must ensure that any block victimized from an L2 segment will be sent to the same coherence controller as the controller that will return the new fill data. (If bit 6 was included in the hash, we could evict an odd block from the L2 segment and replace it with an even block. The protocol described below just won't work that way.)

7.5.6 Outstanding Read CAM (ORC) and Write Back CAM (WBC)

Every read operation in the coherence controller is checked against, and recorded in, the Outstanding Read CAM (*ORC*). The ORC ensures that no new read presented to the coherence controller is allowed to proceed if it conflicts with a read operation already in progress. Similarly, we record all write operations in progress in the WriteBack CAM (WBC). Both ensure that reads and writes to the same block of memory complete in order.

7.5.7 Victim Buffer

We don't implement victim buffers. Since all operations are sequenced through the coherence widgets and the ORC/WBC units, we have no need of "temporary" data storage to cover the ships-passing-in-the-night problems.

7.6 I/O and DMA Transactions

I/O transactions are initiated by Load and Store instructions from the processors, where the physical address refers to I/O space (see Table 7.1). The L2 segment misses (because I/O space addresses are not cached), and the request is presented to the CSW with a target which selects the addressed device (processor, DMA engine, PCIe adapter, Memory Controller, etc. Each L2 segment is permitted to have only one I/O request outstanding at a time; Read requests are completed by the return of read data.

Write transactions are special. Imagine that a processor X initiates a read miss transaction to get a block of data from physical memory. Now imagine that processor Y attempts to write data into a control register on processor X. It is possible that the data for Y's write could arrive at X's bus stop at the same time as the read miss data. We'd have to buffer one of the items. In fact, we could imagine having to buffer several items. That's expensive for an improbable circumstance caused by a low-frequency operation like an I/O write to a processor control register. To simplify the hardware, we require that all data arriving at a bus stop be "pulled" by the recipient. So, when processor Y wishes to write an I/O register in processor X, X will register the write request, and reflect a READ IO request back to processor Y. Processor Y will answer with the data that it wishes to write. (See Table 7.53.)

DMA transactions are initiated by an I/O device connected through the DMA engine or PCIe adapter, and typically reference main memory, but the coherence controller checks each such reference against the master tags. In the event that a read matches, the request is completed by probing the owning cache segment without taking exclusive ownership. When a write matches, the DMA data overwrites the old contents of the cache segment, leaving it valid and modified.

7.7 Coherence Interactions

The data cache segments with each of the processor segments have five states for every block: invalid, shared, exclusive clean, exclusive dirty, and exclusive updated. The cores can change a line from clean or dirty to updated without informing other cache segments.

The cores make only two kinds of requests to the L2 cache: reads and writes. Requests may be qualified in various ways; see Table 7.17.3.

7.7.1 Races

The master tags always change before the local tags, and tag changes are protected from conflict by the OTC. The OTC ensures that any new incoming request is queued while an earlier request for the same block is in progress. When a processor segment evicts a block B from its L2, it must set the block to INVALID in the L2 before issuing any victim write command (or a read command with an implied victim writeback) to the cache switch.

Transfers must notify the coherence controller upon completion, so that any other requests queued for the same block can be cleared. Completion is identified by the Transaction ID code generated by the originator of the request, and is sent by the originator to the coherence controller, which knows the address and former owner of the block.

Because of the sequencing and the dependence chain maintained in the coherence controllers, processor segments need not compare incoming addresses to the L2 writeback buffer. If a victim has been identified and a writeback command has been sent to the coherence controller, the PS *must* return a PROBENOHIT response to the requestor. The requestor will then retry the read command. Again, the dependence chain maintained in the coherence controllers (in the OTC and WBC) ensure that the retried read operation will succeed.

7.7.2 Probes

For most L2 cache accesses, we expect that the master tag will show that no cache had a copy of the requested block, so the block must be obtained from memory. There are, of course, a few exceptions, and for those cases the controller issues *Probe* requests to the cache segment whose tag matches. A probe request contains the physical address of the block in question and indicates to whom the data should be sent. SHARED blocks filling I-stream requests are left in the SHARED state in both requester and responder. Blocks filling I-stream requests will cause ownership to transfer.

It is possible that a probe is on its way to a segment while the block it addresses is being victimized from the segment. In such cases, the responding segment returns PROBENOHIT and the original requester retries the read. The retry, through mechanisms in the coherence controller, is guaranteed to succeed.

A probe response that involves writeback may take many cycles to complete, because it may be necessary to drain the write buffer in the 5kf processor. It is therefore possible to create a backlog of probe requests to a single processor. These are serviced in order of arrival in the L2 segment's command queue.

7.8 Multiprocessor Issues

7.8.1 LL/SC

LL/SC is handled entirely within the ICE9 modifications to the 5kf processor core. When an LL instruction access to the L1 completes, the processor will delay processing of all probe requests from the L2 cache for a programmable number of cycles. Any probes received for the LL target block after this delay will force the SC to fail. For a more complete description of the LL/SC mechanism, see the Section 6.6.10 in the processor chapter.)

7.8.2 Lockstep cache thrashing

Typical applications of the SC 1000 will have many copies of the same program running simultaneously; in some cases that will result in all the processors of a node accessing the same relative location on different pages nearly simultaneously. This would be likely to result in thrashing of the L2 cache if we didn't do something to prevent it, so the cache index is hashed to distribute any page-relative location among many different index values. This does not require a larger tag; the address of a victim can be recalculated by the inverse hash function.

7.8.3 Deadlock Freedom

It is necessary to show that the system is always able to make progress; that requires that there can be no closed cycle of resource dependencies.

An L1 D-cache read can be stalled in the read queue waiting for the ORC, which may report a conflict for the same cache line. The core cannot request another read while there is one outstanding.

The ORC frees dependent transactions when main memory requests complete and when ownership transfers complete.

Memory requests complete with the passage of time. Fills have first priority for use of CSW and L2 cycles.

L2 cache writes (which do not assert transfer) depend only on availability of L2 segment cycles.

Memory writes complete with passage of time; they have no dependencies. I/O writes to PCI space may depend on completion of memory reads or writes.

To ensure that we can drain the write buffer, the processor will be granted a small number of write credits (just enough to keep the pipeline busy) until a probe matches something in the write buffer. At that time, the processor will inhibit instruction issue (as if a SYNC instruction had been found) until the external write buffer is empty, and the external write buffer will make available enough credits to drain the internal write buffer. The interface will separate I/O writes and cached memory writes into separate queues, and update the L2 immediately as the memory writes are issued. This will allow the probe to be satisfied despite delays in I/O service.

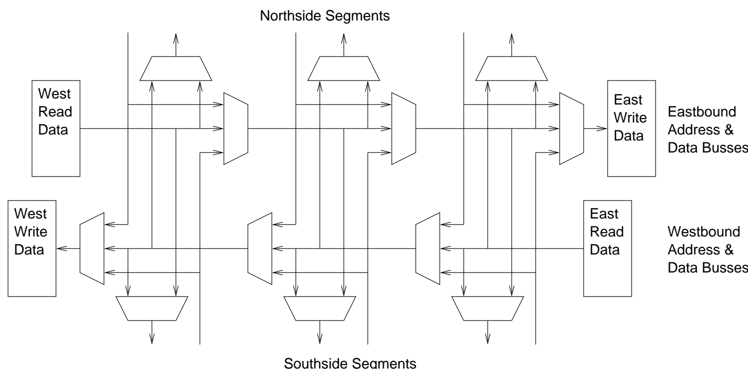
And Wilson is terribly afraid that I'm going to forget to keep transfer requests separate from non-transfer requests; if a transfer request got stuck waiting for a non-transfer request, we could deadlock.

7.9 L2 Segment to Memory Interface

Each segment of the L2 cache includes a block of interface logic by which it communicates with the coherence controllers, the memory and I/O systems, and other segments. Figure 7.4 sketches the interface. The interface consists of two daisy-chain busses, called Evenbound and Oddbound. Each segment decides, whenever it has a request to send, which direction to send it, and watches the Target signals to wait for a cycle in which the bus is free. At the same time, it monitors its own target signal to determine when the bus contents are for it.

Each segment has only one read request outstanding at a time, so there is no danger of receiving data from both memory controllers at once, but it is possible to receive probes simultaneously from both coherence controllers; they must be captured and queued. I/O devices may have multiple outstanding reads, and therefore need the ability to accept two or more responses simultaneously.

Figure 7.4: Memory Bus Interface



7.9.1 Transaction ID

Every command on the Request/Address bus is accompanied by a transaction id, which identifies the originator of the request and uniquely identifies the transaction among the outstanding requests by that originator. There are eight originators: the six processors, the DMA engine, and the PCI-express controller. The DMA and PCI/PMI units may each have up to four reads and four writes outstanding. A processor segment may have an IO read, an IO write, a cache owned-to-shared transfer (WRSTRANS) and a cache fill/replacement outstanding – all simultaneously.

Table 7.1: Memory Bus Port Signals From and To Processor Segment X

Signal Name	Description
psX_csw_CmdAddrTarget_c0a[7:0]	Command/Address Destination
psX_csw_{E/O}CmdAddrReq_c0a	Request for access to command/address bus
csw_psX_CmdAddrGnt_c1a	Grant from switch to PS allowing access
psX_csw_Command_c0a[4:0]	Operation to be performed
psX_csw_CmdAddrTID_c0a[5:0]	Transaction ID for this operation
psX_csw_Addr_c0a[35:3]	Address of cache miss, I/O ref, write, or probe
psX_csw_BMask_c0a[7:0]	Byte mask for I/O commands
psX_csw_Way_c0a	Way select
psX_csw_CmdOwnLock_c0a	See Section 7.8.1.
psX_csw_CmdClearMLAR_c0a	See Section 7.8.1.
psX_csw_DataTarget_c2a[7:0]	Data destination select
psX_csw_{E/O}DataReq_c2a	Request for access to data bus
csw_psX_{E/O}DataGnt_c3a	Grant from switch to PS allowing access
psX_csw_DataTID_c2a[5:0]	Match data to request
psX_csw_DatOwnLock_c2a	See Section 7.8.1.
psX_csw_DatReqLock_c2a	See Section 7.8.1.
psx_csw_DatClearMLAR_c2a	See Section 7.8.1.
psX_csw_ModState_c2a[1:0]	See Section 7.8.1.
psX_csw_HalfMask_c2a[1:0]	Writing 8 bytes, 64 bytes, first 32 bytes, last 32 bytes.
psX_csw_Data0_c2a[72:0]	Doubleword 0,2,4,6 of block (multiplexed)
psX_csw_Data1_c2a[72:0]	Doubleword 1,3,5,7 of block
csw_psX_CmdAddrTID_c1a	Transaction ID for incoming request
csw_psX_CmdAddrValid_c1a	There is an incoming request
csw_psX_Command_c1a[4:0]	Incoming command
csw_psX_Origin_c1a[3:0]	Originating node (for forwarded commands)
csw_psX_Addr_c1a[35:3]	Incoming address
csw_psX_CmdReqLock_c1a	See Section 7.8.1.
csw_psX_BMask_c1a[7:0]	Byte mask for I/O reads and writes
csw_psX_DataTID_c3a[5:0]	Transaction ID for incoming data
csw_psX_DataValid_c3a	If true, incoming data is worth looking at
csw_psX_DataLocked_c3a	Coherence engine found MLAR match for this block from psX.
csw_psX_DataReqLock_c3a	See Section 7.8.1.
csw_psX_DataOwnLock_c3a	See Section 7.8.1.
csw_psX_ModState_c3a[1:0]	See Section 7.8.1.
csw_psX_HalfMask_c3a[1:0]	Writing 8 bytes, 64 bytes, first 32 bytes, last 32 bytes.
csw_psX_Data0_c3a[72:0]	Doubleword 0,2,4,6 of incoming block (muxed)
csw_psX_Data1_c3a[72:0]	Doubleword 1,3,5,7 of incoming block
csw_psX_TIDBusy_c5a[1:0]	A Coherence Widget claims that TID 0 and/or 1 is busy.

(All six processor segments have identical signal ports. Replace "psX" in the above with ps0, ps1... Segments can send a command to either the Even side controller or the Odd side controller as designated by the {E/O} prefix. So, in fact, segment 0 has *two* address/command request signals: ps0_csw_ECmdAddrReq_c0a and ps0_csw_OCmdAddrReq_c0a.) The PCI interface is identical to the PS interface: replace psX in all signal names with pci for this interface.

Table 7.2: Memory Bus Port Signals From and To DMA or PCI Segment

Signal Name	**Description**
dma_csw_CmdAddrTarget_c0a[7:0]	Command/Address Destination
dma_csw_{E/O}CmdAddrReq_c0a	Request for access to command/address bus
csw_dma_CmdAddrGnt_c1a	Grant from switch to PS allowing access
dma_csw_Command_c0a[4:0]	Operation to be performed
dma_csw_CmdAddrTID_c0a[5:0]	Transaction ID for this operation
dma_csw_Addr_c0a[35:3]	Address of cache miss, I/O ref, write, or probe
dma_csw_BMask_c0a[7:0]	Byte mask for I/O commands
dma_csw_Way_c0a	Way select
dma_csw_CmdOwnLock_c0a	See Section 7.8.1.
dma_csw_CmdClearMLAR_c0a	See Section 7.8.1.
dma_csw_DataTarget_c1a[7:0]	Data destination select
dma_csw_{E/O}DataReq_c1a	Request for access to data bus
csw_dma_{E/O}DataGnt_c2a	Grant from switch to PS allowing access
dma_csw_DataTID_c2a[5:0]	Match data to request
dma_csw_DatOwnLock_c2a	See Section 7.8.1.
dma_csw_DatReqLock_c2a	See Section 7.8.1.
dma_csw_DatClearMLAR_c2a	See Section 7.8.1.
dma_csw_ModState_c2a[1:0]	See Section 7.8.1.
dma_csw_HalfMask_c2a[1:0]	Writing 8 bytes, 64 bytes, first 32 bytes, last 32 bytes.
dma_csw_Data0_c2a[72:0]	Doubleword 0 of block (with ECC)
dma_csw_Data1_c2a[72:0]	Doubleword 1 of block
dma_csw_Data2_c2a[72:0]	Doubleword 2
dma_csw_Data3_c2a[72:0]	Doubleword 3
dma_csw_Data4_c2a[72:0]	Doubleword 4
dma_csw_Data5_c2a[72:0]	Doubleword 5
dma_csw_Data6_c2a[72:0]	Doubleword 6
dma_csw_Data7_c2a[72:0]	Doubleword 7
csw_dma_CmdAddrTID_c2a	Transaction ID for incoming request
csw_dma_CmdAddrValid_c2a	There is an incoming request
csw_dma_Command_c2a[4:0]	Incoming command
csw_dma_Origin_c2a[3:0]	Originating node (for forwarded commands)
csw_dma_Addr_c2a[35:3]	Incoming address
csw_dma_BMask_c2a[7:0]	Byte mask for I/O reads and writes
csw_dma_DataTID_c3a[5:0]	Transaction ID for incoming data
csw_dma_DataValid_c3a	If true, incoming data is worth looking at
csw_dma_Data0_c3a[72:0]	Doubleword 0 of incoming block (muxed)
csw_dma_Data1_c3a[72:0]	Doubleword 1 of incoming block
csw_dma_Data2_c3a[72:0]	Doubleword 2
csw_dma_Data3_c3a[72:0]	Doubleword 3
csw_dma_Data4_c3a[72:0]	Doubleword 4
csw_dma_RdTIDBusy_c5a[3:0]	A Coherence Engine claims that TID[x] is currently in flight
csw_dma_WtTIDBusy_c5a[3:0]	A Coherence Engine claims that TID[x] is currently in flight

7.9.2 Target

Every transfer on the Request/Address bus or the Data bus is directed to a specific destination, which may be one of the originating interfaces or one of the two coherence controllers and their associated memory interfaces. When driving the bus, each interface selects either Evenbound or Oddbound direction, depending on the relative positions of source and destination. When responding to a request, the target is decoded from the originator portion of the transaction id. Original requests are always sent to the coherence controller indicated by address bit 6 (should be programmable).

In addition to the Target bits, the coherence controllers can assert Cmd_Bcast in conjunction with all the Target bits to cause all receivers to accept an invalidate command.

```
Target vectors are calculated to have a number of ■1■ bits set equal to the
distance between the sending and the recieving node.  The ■leading■ 1 is
eliminated for the target calculation in all nodes other than the COH.
```

```
targetVectorType bsn2target(fromBSN, toBSN) {
  if(fromBSN is COHO or COHE) {
    return shiftLeft(1, abs(fromBSN - toBSN)) - 1;
  }
  else {
    return shiftLeft(1, abs(fromBSN - toBSN) - 1) - 1;
  }
}
```

Table 7.3: Target Addressing

As shown in Table 7.3, each interface to the Mem Bus generates an 8-bit target mask. The mask determines how many downstream interfaces are expected to forward the data. The interface calculates the differernce between its bus stop number and the destination's bus stop number. It then sets that number of bits (less 1) at the lsb end of the target vector. When the switch grants a bus cycle to an interface, it augments the provided 8 bit target with the request line from the interface. This additional bit is driven downstream as the lsb of the complete (9 bit) target. This allows the downstream node to determine if there is live data on the bus.

7.9.3 Completion

When requestors receive fill data from other caches (that is, from any element other than the memory controller), they notify the coherence controller by sending the transaction id (and possibly other bits). This allows the coherence controller to know when it can release any other request for the same address. Fills from memory can notify the coherence controller directly. Such notice should be timed to allow a cache hit and transfer, rather than initiating another memory request.

7.9.4 CSW Bus Arbitration

The memory bus consists of two sets of separately arbitrated wires (see Table 7.1):

1. Evenbound request/address/data
2. Oddbound request/address/data

Each such set has its own arbitration at each L2 segment; the segment can send if and only if (a) it wants to and (b) there is nothing on the wires from upstream. Arbitration controls use of the entire set of even- or odd-bound wires. Note that read commands optionally transfer a victim, but do not explicitly send the victim address. The coherence controller can determine the victim address from the master tags and way select.

The coherence controller may prevent any bus stop from winning arbitration in order to prevent overflow of the DDR controller request buffers. This merely imposes a delay in time, but may not create deadlock opportunities.

7.9.4.1 Fairness

How do we prevent a segment being locked out of bus access by traffic from upstream? First, we should note that we can do all kinds of calculations that show that we'll never really tax the capacity of the CSW or DDR controllers. And then we'd find a chip that hung because we taxed the capacity of the CSW or DDR controllers. So, the arbitration protocol prevents complete lockout by rationing access to the CSW when there is contention.

If a bus stop (say the DMA engine) initiates a request in cycle 0, it will find out in cycle 1 if it won the bidding. Assume that it wins. It may have triumphed over some other downstream bus stop X. (Note that in the even-bound direction, almost everybody is downstream of the DMA engine.) In this case, the DMA engine will not win further arbitration for the bus until EVERY downstream bus stop that lost to the DMA bid is eventually granted access to the CSW chain. This is implemented completely within the CSW arbitration logic.

7.9.4.2 Worst Case Traffic Analysis

Every request to the memory arrays requires one 4ns cycle of the Memory Bus, and eight edges of the memory's DQ bus. We're designing for DQ bus clock rates up to 400 MHz, so 8 edges take 10ns; thus the memory bus cannot be more than 40% saturated by main memory traffic. In addition, inter-cache transfers can occur concurrently with main memory access. Each such access encounters a minimum latency of 12 4ns cycles, so the maximum possible bus loading is 6 requestors/12 cycle latency = 50%. The worst loading at any point on the bus is less than this because the requests have to be distributed among many L2 segments to be requested and serviced that quickly, with the result that the bus isn't occupied for its full length, and the interface in question will be able to share at least some of the used cycles.

We also have to account for the DMA engine and PCI-express controller, each of which can have four requests outstanding at any time, but only two of them can be to the same memory controller, and very few of which result in inter-cache transfers.

7.9.5 CSW Queuing of Commands and Data

At each CSW bus stop, one module can inject commands or data onto the Even or Odd memory bus, and the CSW can deliver commands or data to the module. Incoming commands may arrive two per cycle (one from each direction), but the bus stop interface can only transmit one of those commands into the module per cycle. The CSW contains queues in each bus stop to handle cases where commands arrive too fast. Data can also arrive from both directions at once, if a module ever requests multiple data transfers at a time. The processor segments limit themselves to one data request at a time, so no queuing is required in their bus stops, but the DMA and PCI can make multiple outstanding data requests, so their bus stops require data queues. The depth requirements for each queue are analyzed below, for each type of bus stop.

To know how deep the command and data queues should be, we must identify a worst case number of commands that could arrive at this bus stop, and consider how quickly the module can consume the transactions as they are coming in. A bus stop could receive one command per TID in the system: 12 processor TIDs, 8 DMA TIDs, and 8 PCI TIDs. [NOTE: this analysis assumes that INTs consume a TID, and a block will not send another INT until a DONE response comes back.] In the worst case, these 28 commands could arrive in 14 consecutive cycles, half coming from the even side and half coming from the odd side. Half of them can be consumed by the module, while the other half must be queued. So the command queue for each bus stop must be 14 commands deep. For the data queues, the answer depends on the number of outstanding data transactions that the module can produce. The processor segment is careful to only allow one data transaction at a time, while DMA and PCI can have 4 reads outstanding plus some number of WTIOs.

Table 7.4 summarizes these results.

The different requirements for bus stops leads to the need for several bus stop variants. The command side of all bus stops are all copies of the same module (CswPca), whose queue structure is described in Figure 7.5. Commands from even and odd sides are queued if necessary, and the bus stop delivers one command at a time to the target module. The processor needs no data queue. The PCI bus stop queues data from even and odd sides, and delivers it to the PCI at a rate of two doublewords per cycle (Figure 7.6). The DMA bus stop queues data from even and odd sides, and delivers it to the DMA at a rate of eight doublewords per cycle (Figure 7.7).

7.9.6 Transfer order

Data transfers on the CSW are ordered to ensure a fixed pipeline timing for each section of the bus, while delivering cache miss data to processors starting with the requested word first, and keeping aligned 16-byte units

Bus Stop	Cmd/Data	Max Arriving, Worst Case	Number Consumed	Queue Depth Needed
All	Cmd	12 processor TIDs (probes) + 8 DMA TIDs + 8 PCI TIDs Total: 28 commands in 14 cycles	14 in 14 cycles	14
Processor	Data	1 read response ....or.... 1 WTIO data word, but never both at once	1 every 4 cycles	none
PCI	Data	4 read responses + 2 WTIO data words Total: 6 transfers in 3 cycles (The PCI bus stop supports two modules which can each do WTIOs.)	1 every 4 cycles	5
DMA	Data	4 read responses + 1 WTIO data word Total: 5 transfers in 3 cycles	3 in 3 cycles	2

Table 7.4: Queue Depth Requirements for CSW Bus Stops

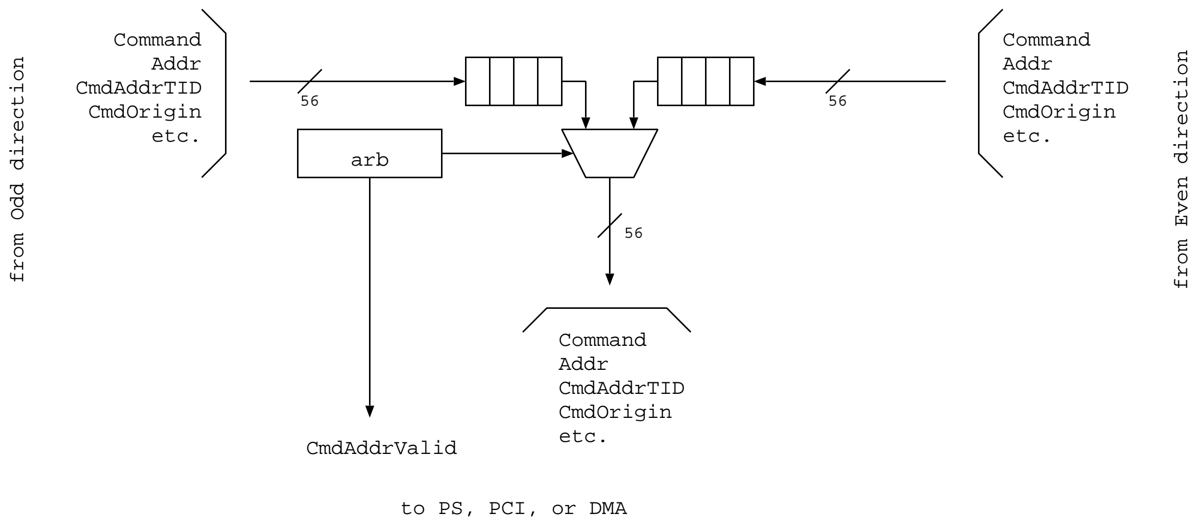


Figure 7.5: CSW Queues for CmdAddr Requests

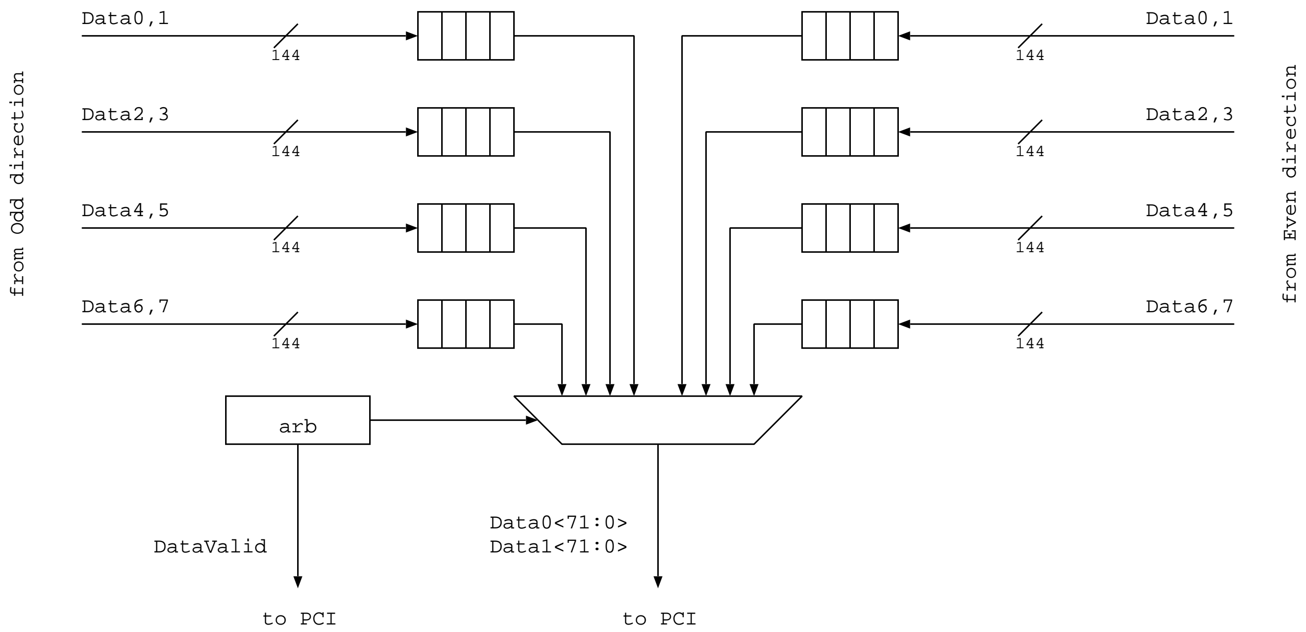


Figure 7.6: CSW Data Queues for PCI Bus Stop

Figure 7.7: CSW Data Queues for DMA Bus Stop

together. Note that address bit 3 is ignored, and setting bits 4 and/or 5 result in exchanging the order of halves of the block.

Table 7.5: Transfer sequence as a function of address

Address	Data0, Data1	Data2, Data3	Data4, Data5	Data6, Data7
x..x00	07:00, 0F:08	17:10, 1F:18	27:20, 2F:28	37:30, 3F:38
x..x08	07:00, 0F:08	17:10, 1F:18	27:20, 2F:28	37:30, 3F:38
x..x10	17:10, 1F:18	07:00, 0F:08	37:30, 3F:38	27:20, 2F:28
x..x18	17:10, 1F:18	07:00, 0F:08	37:30, 3F:38	27:20, 2F:28
x..x20	27:20, 2F:28	37:30, 3F:38	07:00, 0F:08	17:10, 1F:18
x..x28	27:20, 2F:28	37:30, 3F:38	07:00, 0F:08	17:10, 1F:18
x..x30	37:30, 3F:38	27:20, 2F:28	17:10, 1F:18	07:00, 0F:08
x..x38	37:30, 3F:38	27:20, 2F:28	17:10, 1F:18	07:00, 0F:08

7.10 Detailed Interface and Block Descriptions

7.10.1 The Normal Flow Of Events, Hazards, and General Ordering Cases

Almost all the mischief that can happen in a cache/memory system surrounds the handling and ordering of reads. Writes almost take care of themselves. So, I'll attempt to explain the operation of the coherence widget by looking at the way read operations interact with other read operations and write operations and the distributed L2 cache.

Note that we're talking about a system with a split bus – that is, a read transaction is split into a read request for address A from processor X (which we note as Read(X,A)), and a data response which we'll write as ReadData(X,A,D) if we ever need to. Similarly, we break write operations into Write(X,A) and WriteData(X,A,D) since the data may be delivered many cycles after the corresponding address.

The tables below, one for each kind of transaction, describe the sequence of events to carry out the transaction. When a unit transmits a command into the cache switch, we denote the operation as CMD(C,U,T,A,W,L,O) where

C is the command being transmitted.

U is the target unit to which this command is being sent. It is one of P0, P1, P2, P3, P4, P5, PCI, DMA, COHE, or COHO.

T is the transaction ID. Tx designates a transaction ID that contains the unit for unit X in its upper bits.

A is the relevant address, or the value to be driven onto the address bus.

W is the L2 cache way that will hold the returned data. W is not always relevant to a command, in those cases, it ommitted.

L indicates that the block in question had an outstanding load/link operation registered on it by the sending processor. L is not always relevant to a command, in which case it will be omitted.

O indicates an "originator" field. This is almost always optional. When used it will be represented as ORIGIN=value.

The data portion of the transaction will be represented as DATA(U,T,D,s) where U and T are as described above, and

D is the data to be transfered, either 8 bytes, 32 bytes, or 64 bytes.

s is the size and placement of the transfer. It indicates that the block is either 8 bytes long, 32 bytes long, starting with doublewords 0 and 1, 32 bytes long starting with doublewords 4 and 5, or 64 bytes long.

The text below refers to the "command bus" or the "data bus." We don't really have "buses" in the chip, instead we have pipelined-multitap-multiplexed-daisychains, but "bus" is a little easier on the eyes. For purposes of understanding the flow of the transactions, "bus" is a reasonable approximation of what we're implementing. For more detail, see 7.15.2.

7.10.2 Transaction Steps and the CSW Buses

The two bus events described above CMD() and DATA() require signals to be sequenced over several cycles or pipeline stages on the CSW ports. For example, CMD(RDEX,COHE,0x6,0x2badbeef0, 1), meaning "Read and acquire Exclusive Ownership from the Even Coherence widget, block 0x2badbeef0. Register the new owner (processor 3) as caching this block in way 1" appears on the processor port to the bus as shown in Figure 7.8. The sequencing for the event DATA(D[7:0], Px, TID, 64) is shown in Figure 7.9. Half block transfers may be to either the first 32 bytes of a 64 byte block, or the second. These two transfers are shown (from the processor's view) in Figures 7.10 and 7.11. Finally, 8 byte transfers (used for I/O operations) are described in Figure 7.12.

The DMA engine interface to the CSW is different from the other interfaces because it has eight 72-bit buses in each direction instead of two. Figure 7.13 shows how the data is staged onto Data0-1 in one cycle, then Data2-3 in the next, and so on. The DMA can send and receive back-to-back transactions on the CSW.

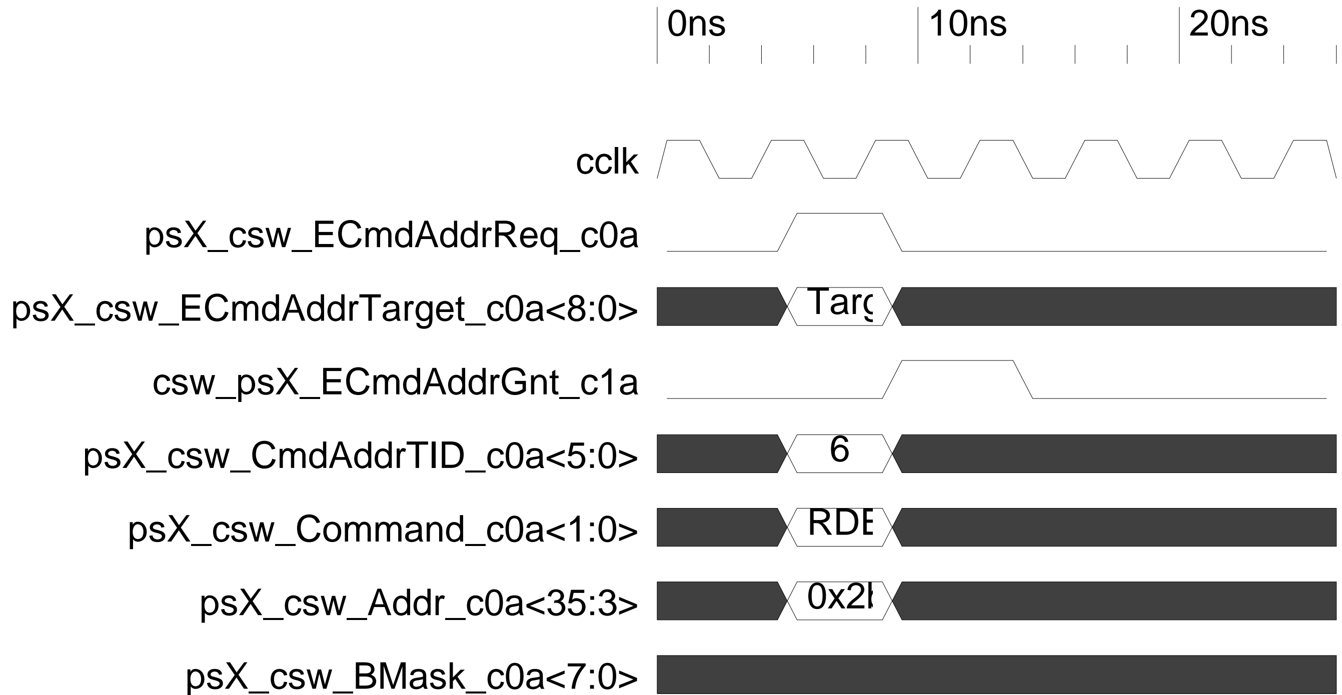


Figure 7.8: Signalling Sequence for CMD(RDEX, COHE, 0x6, 0x2badbeef0, 1)

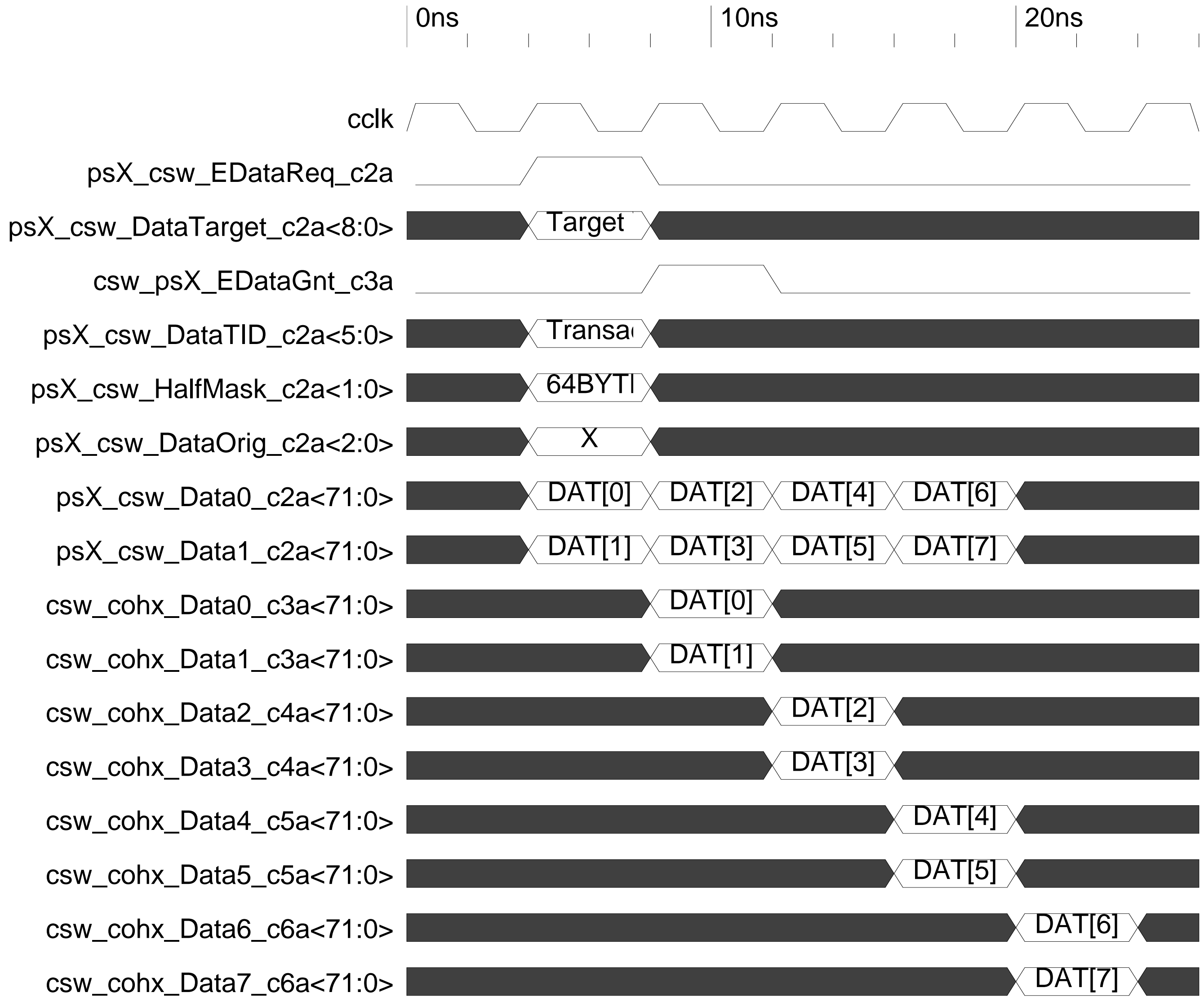


Figure 7.9: Signalling Sequence for DATA(DAT[7:0], Px, TID, 64)

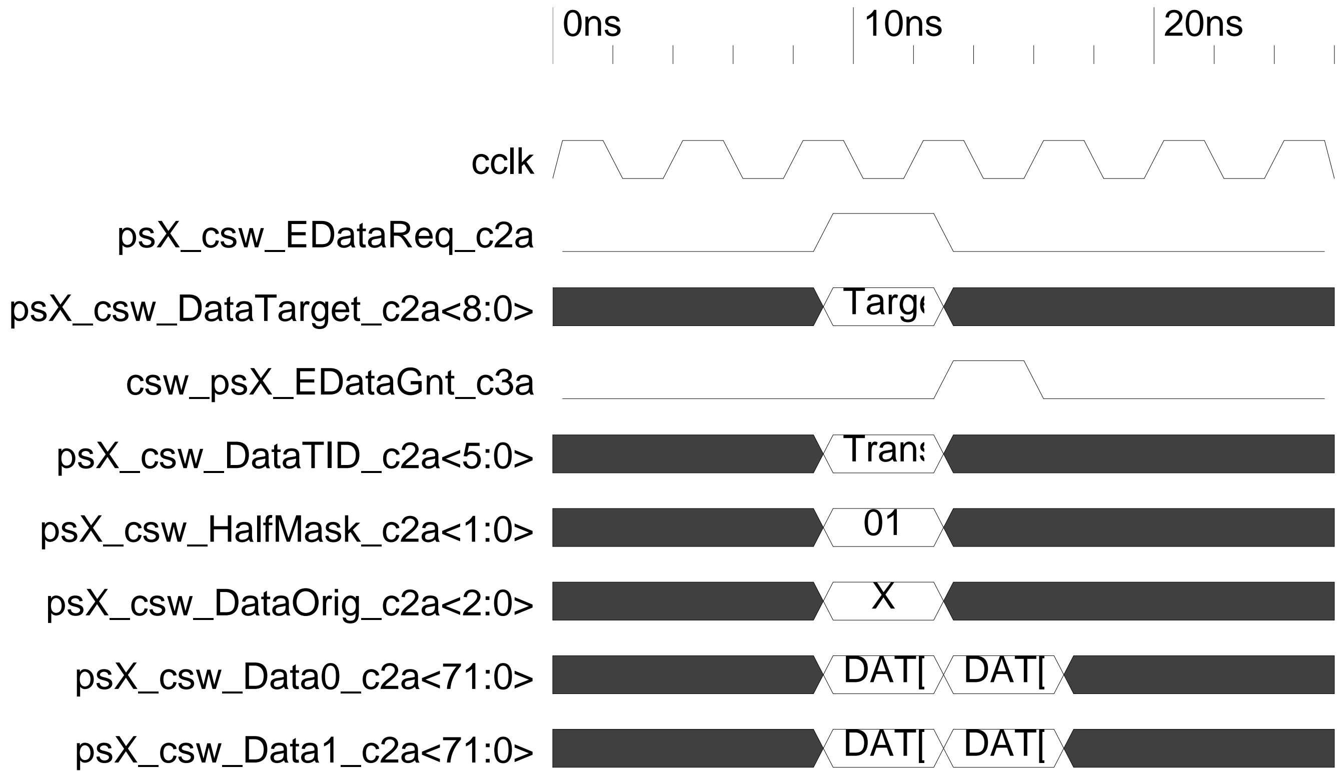


Figure 7.10: Signalling Sequence for DATA(DAT[3:0], Px, TID, 32F)

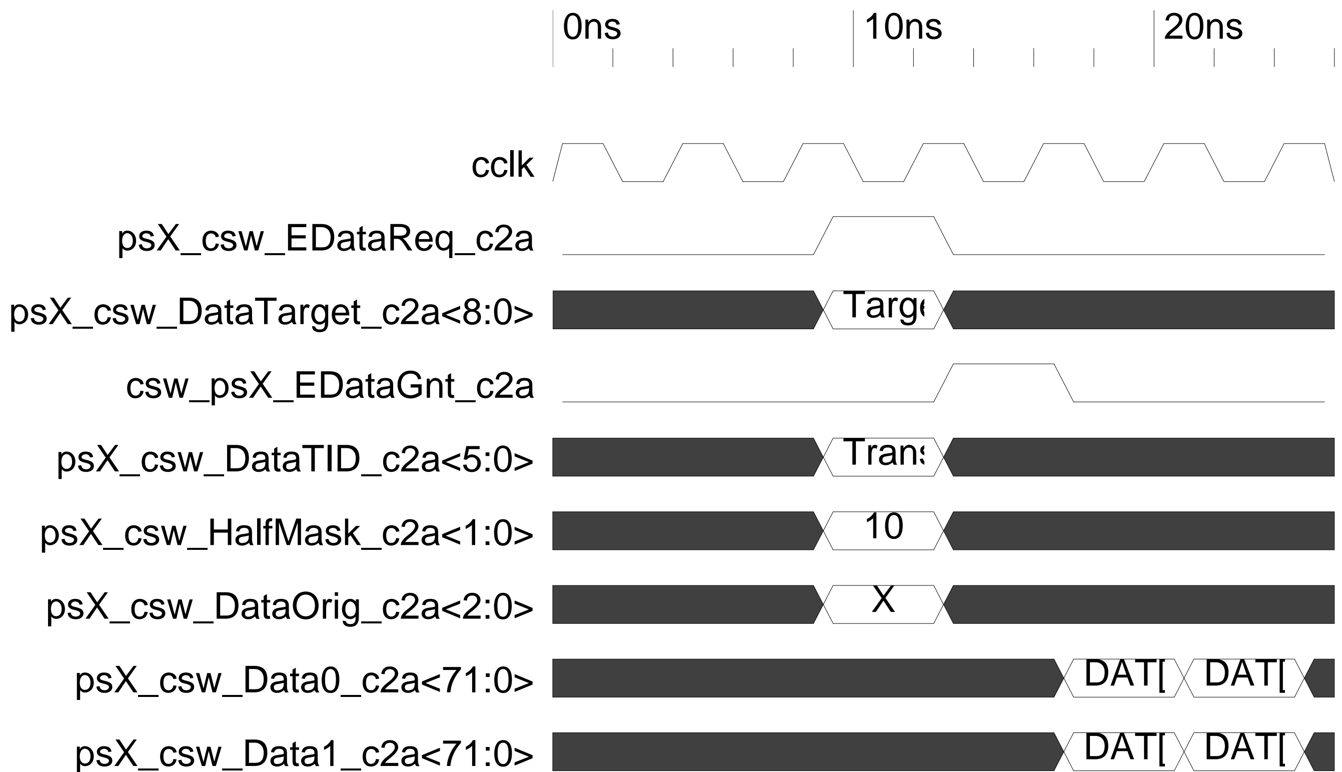


Figure 7.11: Signalling Sequence for DATA(DAT[3:0], Px, TID, 32S)

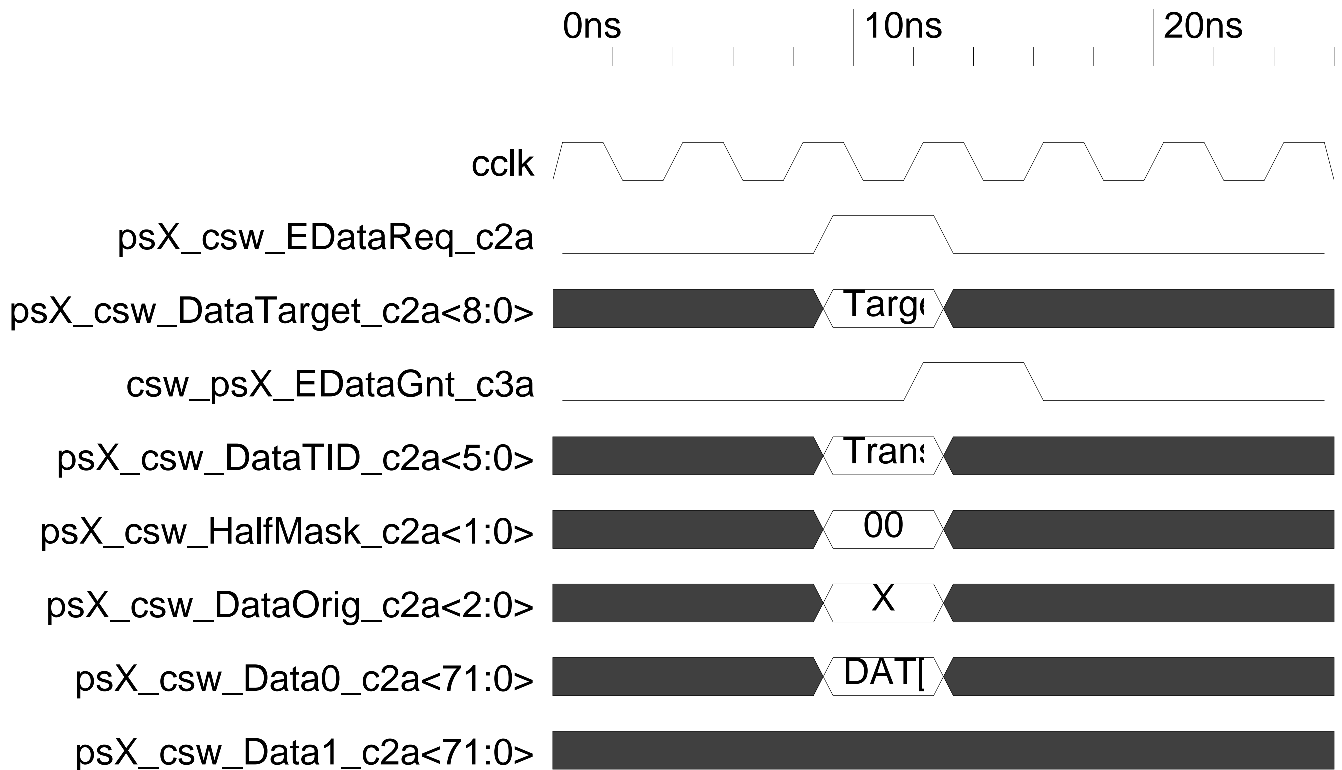


Figure 7.12: Signalling Sequence for DATA(D, Px, TID, 8)

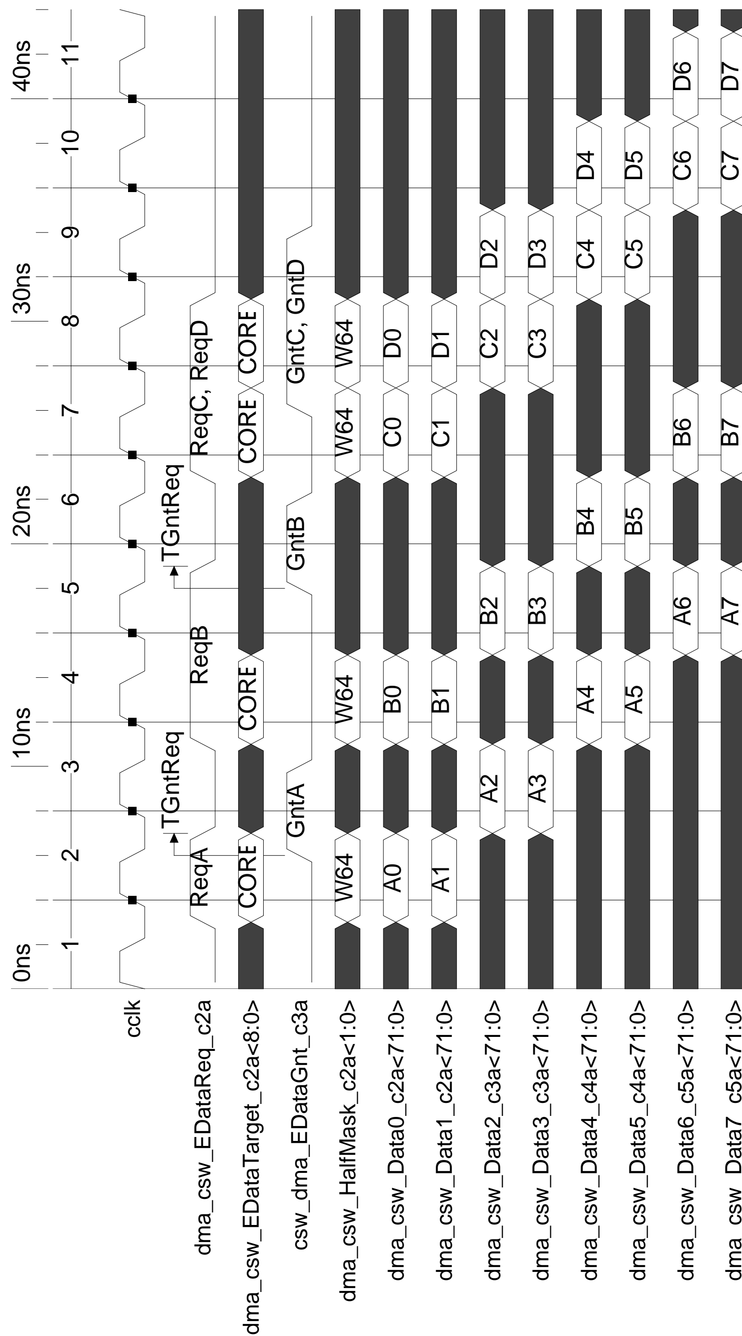


Figure 7.13: Signalling Sequence for DATA(DAT[7:0], DMA, TID, 64) from the DMA Engine
Transaction A is granted in the following cycle (the fastest possible grant). Transaction B is granted after one stall cycle. Transactions C and D are requested and granted back-to-back. Note that the CSW samples all values relative to the request cycle, not the grant, and the CSW stores the content of the request until the request is granted. The DMA is not required to hold the data during stalls. (Other CSW bus stops have other rules.)

7.10.3 The Outstanding Read CAM and the Write Back CAM

The ICE9 L2 cache system supports six processors, and DMA engine and a PCI express widget and can field up to 28 transactions at any one time. Each of the six processors can have one read and one write transaction outstanding at a time. The DMA engine and PCI widget can have four reads and four writes *each* at a time.

We want to maintain memory ordering to at least an intuitive degree. That is, processors never see "time going backwards." I could go on for a bunch of pages about strong consistency vs. weak consistency. Suffice it to say, we want memory ordering semantics that are the same as we implemented with MIPS multiprocessors. Whatever that is.

We make sure that the L2 system doesn't re-order reads and writes to the same "block" (32 bytes) relative to each other by chaining operations to the same block together in the outstanding read CAM and the write back CAM. We cover this in a fair amount of detail in the sections on read and write ordering hazards, below. In the transaction flows below we identify several operations on the ORC and WBC.

ORC_Reg(X,A,T) store A – the block address, and T – the transaction ID as the keys in the ORC. Store X in the requester field. ORC_Reg also remembers whether the address for this entry had matched against any other ORC when it was first looked up. (This is used in the EXCLUSIVE to SHARED transition.) Such entries have their "HEAD_OF_LIST" bit set. All others have this bit cleared.

ORC_Check(A) Lookup A in the ORC. Match only against ORC entries who's Xd, Ad, Td, Op fields are empty. (*i.e.* those that have no dependents)

ORC_CheckS(Tx) Lookup transaction ID Tx in the ORC. This is used by the WRSTRANS operation.

ORC_Dep(Ty,Xd,Ad,Td,Op) find the entry matching transaction ID Ty and store a dependent operation from node Xd, using block address *offset* Ad, transaction ID Td, and Op.

ORC_Rel(T) find the entry matching TID T, If the Xd field is not null, then there was a dependent read or block write operation queued up behind this read. Launch the dependent operation. Clear the valid bits for the matching CAM entry. (Release the entry.)

WBC_Reg(X,A,T) store A – the block address, and T – the transaction ID as the keys in the WBC. Store X in the requester field.

WBC_Check(A) Lookup A in the ORC. Match only against ORC entries who's Xd, Ad, Td, Op fields are empty. (*i.e.* those that have no dependents)

WBC_GetAddr(T) Lookup T in the ORC. Return the value for A at the matching location. (This is how we retrieve the write address that goes along with a block of data. Note that data is sent several cycles after the write address arrives.)

WBC_Dep(Ty,Xd,Ad,Td,Op) find the entry matching transaction ID Ty and store a dependent operation from node Xd, using block address *offset* Ad, transaction ID Td, and Op.

WBC_Rel(T) find the entry matching TID T, If the Xd field is not null, then there was a dependent read or block write operation queued up behind this read. Launch the dependent operation. Clear the valid bits for the matching CAM entry. (Release the entry.) Note that WBC_Rel is triggered on *completion* of a write with respect to the DDR controller or – in the case of forwarded writes – completion signalled by a BWTDONE. See Section

We also perform a few operations on the L2 master tags.

TAG_Check(A) Lookup A in the L2 master tag arrays (one for each of the 6 processor/cache segments). Return the state and a list of matching entries.

TAG_Update(P,A,W,S) Create an L2 master tag entry in the tag array for processor P, in way W, with address A. Set the state to S. State is one of EX (exclusive), SH (shared), or IN (invalid).

TAG_Victim(A,W) Return the address of the victim block for the L2 tag array at the index derived from A for way W. (For operations that include an implicit victim write, we need the address of the victim block.)

7.10.3.1 The ORC

The ORC is indexed as a CAM with the Address of interest as a key. It can also be directly indexed by TID. Each entry in the table contains eleven fields

Valid True if this entry represents a currently outstanding memory read transaction

AddrTag The block address of the corresponding transaction

Last True if this is the last memory read or write operation posted for AddrTag

Excl True if the block was in the EXCLUSIVE state when the operation was first registered in the ORC

Shr True if the block was in the SHARED state when the operation was first registered in the ORC

Own The is the processor identifier of the current owner of the block if the block was SHARED and some processor segment claims that it is willing to source the data. (The owner, if it exists, is the last to acquire the block. It is possible, however, that the last acquirer has evicted the block. In this case, the OWN field points to a non-existedn processor segment (0xf).

DepTID The TID of an operation that was registered as a dependent on this entry. Valid only if Last is false.

DepCmd The command for the dependent operation. Valid only if Last is false.

DepAddr The low bits of the address of the dependent operation.

DepOrg The originator of the dependent operation.

SrcCmd The command that created this entry in the ORC.

7.10.3.2 The WBC

The WBC is indexed as a CAM with the Address of interest as a key. It can also be directly indexed by TID.

Valid True if this entry represents a currently outstanding memory read transaction

AddrTag The block address of the corresponding transaction

Last True if this is the last memory read or write operation posted for AddrTag

Winv True if this entry corresponds to a writeback (WINV) or victimization (RDV, RDSV)

Shr True if this entry was in the SHARED state when it was created.

LowBits The low bits of the address for the dependent operation. Valid only if Last is false.

DepTID The TID of an operation that was registered as a dependent on this entry. Valid only if Last is false.

DepCmd The command for the dependent operation. Valid only if Last is false.

DepOrg The originator of the dependent operation.

DepOwn The owner of the block when the dependent command was registered on this entry.

7.10.4 Transaction Flows

7.10.4.1 D-Stream Read to a Non Resident Block

This is the simplest case, so we start with that. Assume that processor X launches a load operation that misses on block A. The operation may displace a victim block. If it does not, the operation proceeds as a simple read, shown in Table 7.6. If a victim write back is required, the operation is described in Table 7.7.

Note that during the command processing phase of the transaction (cycles 2 and 3) the address is first looked up in the master tags, the writeback CAM (WBC) and the outstanding read CAM (ORC). In the second of the two cycles, we update the tags, the WBC, and the ORC. In the latter case, the update to the CAM array occurs at the start of the cycle, so comparisons to the new CAM entries can occur immediately. The tag arrays, however,

Cycle	PX Action	COH Action	Comment
1	CMD(RDEX,COHn,Tx,A,W)		
2		TAG_Check(A) - no hit found. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR Controller and queue for DDR Read operation.	
3		ORC_Reg(PX, A, Tx) TAG_Update(PX, A, W, EX)	
N		DATA(X,Tx,D) – return Data to Px. ORC_Rel(Tx)	Data is returned in “best word first” order. Px can now launch a new read operation as soon as the first data word arrives.
N+1	Store D in L2/L1.		

Table 7.6: D-Stream Read to a Non Resident Block: No Victim Writeback

are implemented as RAMs and so we must implement a comparison bypass to allow two back to back operations on the same block address to work properly.

Note that the difference between an RDEX and RDV is entirely found in the writeback operation starting with the WBC_Reg in cycle 3, and including the data write cycles beginning in cycle M. Writebacks never stall. That is, the result of tag lookups in the L2 tags, WBC, or ORC has no effect on the writeback or its time of arrival. For this reason, we will show only a few examples of the writeback version of the transaction flows. (For a discussion of the only really interesting thing that can happen to a writeback, see the description of victim writeback collisions against BWT operations in Section 7.10.4.19.)

Cycle	PX Action	COH Action	Comment
1	CMD(RDV,COHn,Tx,A,W)		
2		TAG_Check(A) - no hit found. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR Controller and queue for DDR Read operation. Av = TAG_Victim(A,W)	If a WBC hit is found here, it must be against a BWT.
3		ORC_Reg(PX, A, Tx) WBC_Reg(PX, Av, Tx) TAG_Update(PX, A, W, EX)	We remember that there is a write outstanding since the data may not arrive for some time. The WBC allows us to buffer the write address to send along with the data, and to protect against "ships passing in the night." See Section 7.10.4.17. If Av matches an outstanding BWT, then we write Av = NULL in the WBC.
M	DATA(COH,Tx,Dw) or CMD(WBCANCEL,Tx)		Cycle M may be coincident with cycle 3.
M+1		Data or WBCAN arrives at COH. Aw = WBC_GetAddr(Tx) Send Aw along with the data Dw to the DDR controller. WBC_Rel(Tx)	If Aw is NULL, then this write-back was killed by an intervening BWT request.
N		DATA(X,Tx,Dr) ORC_Rel(Tx)	Data is returned in "best word first" order. Px can now launch a new read operation as soon as the first data word arrives.
N+1	Store Dr in L2/L1.		

Table 7.7: D-Stream Read to a Non Resident Block – With Victim Writeback

Cycle	PX Action	COH Action	DEV Action	Comment
1	CMD(RDEX,COHn,Tx,A,W)			
2		TAG_Check(A) - no hit found. WBC_Check(A) - no hit found. Tv = ORC_Check(A) - HIT! Send A to DDR Controller and queue for DDR Read operation.		If the TAGS are all clear, then the read that we're depending on is a BRD to an uncached location from the DMA engine or PCI.
3		Shootdown A in DDR controller. ORC_Reg(PX, A, Tx) TAG_Update(PX, A, W, EX) ORC_Dep(Tv, PX, A, Tx)		Register our dependence on the earlier read operation.
M		DATA(DEV,Tv,Dr)		Data is returned by the DDR controller. It is not possible for this sequence to end with a forwarded read acknowledged by DMA/PCI (PRBDONE).
M+1		PX, Tx, A = ORC_Rel(Tv)	Dev gets data	
M+2		Send address A to DDR controller.		
M+3	Continue at step N in Table 7.6			

Table 7.8: D-Stream Read to a Non Resident Block – Hit on Oustanding Read CAM.

Cycle	PX Action	COH Action	DEV/PY Action	Comment
1	CMD(RDEX,COHn,Tx,A,W)			
2		TAG_Check(A) - no hit found. Tv = WBC_Check(A) - HIT! ORC_Check(A) - no hit. Send A to DDR Controller and queue for DDR Read operation.		If the TAGS are all clear, then the write that we're depending on either a victim writeback or a BWT to an uncached location.
3		Shootdown A in DDR controller. ORC_Reg(PX, A, Tx) TAG_Update(PX, A, W, EX) WBC_Dep(Tv, PX, A, Tx)		Register our dependence on the earlier write operation.
M			DATA(X,Tv,Dr)	Data is returned to the DDR controller.
M+1		PX, Tx, A = WBC_Rel(Tv)		
M+2		Send address A to DDR controller.		
M+3	Continue at step N in Table 7.6			

Table 7.9: D-Stream Read to a Non Resident Block – Hit on Write Back CAM.

7.10.4.2 D-stream Read to a Cached Block

This is where things get interesting. Consider again the case of a processor X reading block A. In this case, we assume that block A is already resident in some other cache – processor Y for example. Our cache coherence scheme allows a block to be in one of three states: INVALID, EXCLUSIVE, or SHARED. (The SHARED state is implemented for i-stream cache blocks only. This section will describe accesses to a block that is in the EXCLUSIVE state or the INVALID state. For D-stream accesses to blocks in the SHARED state, see Section 7.10.4.5.) In the first case, described in Table 7.10, processor X does not require a victim write back (block A is replacing an INVALID, SHARED, or EXCLUSIVE-CLEAN block). In the second case, described in Table 7.11, processor X must write back a victim block.

Cycle	PX Action	COH Action	PY Action	Comment
1	CMD(RDEX, COHn, Tx, A, W)			
2		TAG_Check(A) - return PY, EX WBC_Check(A) - no hit found. ORC_Check(A) - no hit found Send A to DDR controller and queue for DDR read operation.		
3		CMD(PRBWIN,PY,Tx,A) TAG_Update(PX, A, W, EX) TAG_Update(PY, A, W, IN) ORC_Reg(PX, A, Tx) Send “shootdown” signal to DDR to cancel DDR read of A.		
L			Look up A in L2 tags. Find a hit. Send A to L1 for probe/writeback.	
L+1			Copy data from dirty 32 byte blocks from L1 into 64 byte L2 block (update if the L1 entry was dirty)	
M			DATA(PX,Tx,D,d) – return data to PX. d is true if block A was EXCLUSIVE-DIRTY.	
M+1	Receive data from bus, write to L2/L1. Set to EXCLUSIVE-DIRTY if d was true. Set to EXCLUSIVE-CLEAN otherwise. New read operation can be launched as soon as the first 128 bits of data arrives.			
M+2	CMD(PRBDONE, COH, Tx, addr=0)			
M+3		ORC_Rel(Tx)		

Table 7.10: D-Stream Read of Cached Data – No Victim Writeback

Cycle	PX Action	COH Action	PY Action	Comment
1	CMD(RDV,COHn, Tx,A,W)			W is the way that we'll displace and the target way for A.
2		TAG_Check(A) - return PY, EX WBC_Check(A) - no hit found. ORC_Check(A) - no hit found Av = TAG_Victim(A, W) Send A to DDR controller and queue for DDR read operation.		
3		CMD(PRBWIN,PY,Tx,A) TAG_Update(PX, A, W, EX) TAG_Update(PY, A, W, IN) ORC_Reg(PX, A, Tx) WBC_Reg(PX, Av, Tx) Send "shootdown" signal to DDR to cancel DDR read of A.		
L			Look up A in L2 tags. Find a hit. Send A to L1 for probe/writeback.	L may be as early as cycle 3, but there may be queueing delay at PY's command input.
L+1			Copy data from dirty 32 byte blocks from L1 into 64 byte L2 block (update if the L1 entry was dirty)	
M	DATA(COHn,Tx,Dw) – write-back victim block or CMD(WBCANCEL,Tx)			Cycle M may occur as early as cycle 3. This activity may run in parallel with other parts of this transaction.
M+1		Aw = WBC_GetAddr(Tx) Send Aw along with the data Dw to the DDR controller. WBC_Rel(Tx)		
N			DATA(PX,Tx,D,d) – return data to PX. d is true if block A was EXCLUSIVE-DIRTY.	
N+1	Receive data from bus, write to L2/L1. Set to EXCLUSIVE-DIRTY if d was true. Set to EXCLUSIVE-CLEAN otherwise. New read operation can be launched as soon as the first 128 bits of data arrives.			
N+2	CMD(PRBDONE,COH,Tx,addr=0)			
N+3		ORC_Rel(Tx)		

Table 7.11: D-Stream Read of Cached Data – With Victim Writeback

At times, the PWIN arriving at PY will result in PY finding that the data is no longer in its cache. (PY can autonomously evict an EXCLUSIVE block that is clean, without informing the COH. (This can also happen because of a race between a victimization by PY and a read by PX. See Section 7.10.4.17.) In this case, PY, upon receiving the PWIN command will send a PRBNOHIT command to PX with the original TID. PX will then requeue the Read operation as a REREAD(X,A) and the transaction will proceed as shown in Table 7.12. The table picks up the transaction at cycle L.

Cycle	PX Action	COH Action	PY Action	Comment
L			Look up A in L2 tags. It misses.	
L+1			CMD(PRBNOHIT, PX, Tx, addr=0)	
K	CMD(RDEXR, COHn, Tx, A)			This is a ReadExclusive Retry command. K could be as early as L+2.
K+1		Send A to DDR controller.		
R		Receive read data from DDR OTC_Rel(Tx)		R may be many cycles after L+3.
R+1		DATA(PX, Tx, Dr)		
R+2	Receive data from bus, write to L2/L1. Set to EXCLUSIVE-CLEAN.			

Table 7.12: Forwarded D-Stream Read Misses in Probed Cache

Cycle	PX Action	COH Action	PY/DEV Action	Comment
1	CMD(RDEX, COHn, Tx, A, W)			
2		TAG_Check(A) - return PY, EX WBC_Check(A) - no hit found. Tv = ORC_Check(A) - HIT! Send A to DDR controller and queue for DDR read operation.		ORC hit is either on PY doing the initial read that fills this block in PY or on a BRD to PY. (Otherwise, the state wouldn't be PY EXCLUSIVE.)
3		TAG_Update(PX, A, W, EX) TAG_Update(PY, A, W, IN) ORC_Reg(PX, A, Tx) ORC_Dep(Tv, Px, A, Tx) Send "shootdown" signal to DDR to cancel DDR read of A.		Register this transaction as dependent on an earlier read transaction with TID = Tv.
K		DATA(DEV, Tv, D)	CMD(PRBDONE, COHn, Tv, addr=0)	Either read data is supplied by DDR to PY or PY completed via an inter-cache transfer.
K+1		Px, Tx, A, Py= ORC_Rel(Tv)		
K+2		CMD(PRBWIN, PY, Tx, A)		
K+3	Continue with step L in Table 7.10			

Table 7.13: D-Stream Read of EXCLUSIVE Block – ORC Hit

Cycle	PX Action	COH Action	DEV Action	Comment
1	CMD(RDEX, COHn, Tx, A, W)			
2		TAG_Check(A) - return PY, EX ORC_Check(A) - no hit found. Tv = WBC_Check(A) - HIT! Send A to DDR controller and queue for DDR read operation.		WBC hit against a block write operation to processor PY.
3		TAG_Update(PX, A, W, EX) TAG_Update(PY, A, W, IN) ORC_Reg(PX, A, Tx) ORC_Dep(Tv, Px, A, Tx) Send “shootdown” signal to DDR to cancel DDR read of A.		Register this transaction as dependent on an earlier read transaction with TID = Tv.
K			CMD(BWTDONE, COHn, Tv, addr=0)	
K+1		Px, Tx, A, Py= WBC_Rel(Tv)		
K+2		CMD(PRBWIN, PY, Tx, A)		
K+3	Continue with step L in Table 7.10			

Table 7.14: D-Stream Read of EXCLUSIVE Block – WBC Hit

7.10.4.3 I-stream Read to a Non Resident Block

ICE9 supports cache coherency via an exclusive writer model. That is, the cache does not support a "shared-update" operation where one processor is able to write a few bytes through and update cache blocks in other processors. It isn't that we don't like shared-update protocols, it's just that such protocols are really hard to verify and hard to retrofit to a processor pipeline that was built for a simpler model.

But we *do* want to share I-stream data among the caches. So, we implement a SHARED state in the cache. Blocks in the SHARED state can't be written. They only get into the shared state as the result of an I-stream L1 cache miss.

Cycle	PX Action	COH Action	Comment
1	CMD(RDS,COHn,Tx,A,W)		Istream read, into way W for L2
2		TAG_Check(A) - Find no matches. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller	
3		ORC_Reg(PX, A, Tx). TAG_Update(PX, A, W, SH)	
N		DATA(PX,Tx,Di) ORC_Rel(Tx)	DDR returns data.
N+1	Receive data from bus, write into L2 and L1 ICache. Set state to SHARED.		

Table 7.15: I-Stream Read to a Non Resident Block

Cycle	PX Action	COH Action	Comment
1	CMD(RDSV,COHn,Tx,A,W)		
2		TAG_Check(A) - no hit found. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller Av = TAG_Victim(A, W)	
3		ORC_Reg(PX, A, Tx) WBC_Reg(PX, Av, Tx) TAG_Update(PX, A, W, SH)	
M	DATA(COHn,Tx,Dw) or CMD(WBCANCEL,Tx)		This is the victim writeback. M may occur as early as cycle 3.
M+1		Aw = WBC_GetAddr(Tx) Send Aw along with the data Dw to the DDR controller. WBC_Rel(Tx)	
N		DATA(PX,Tx,Di) ORC_Rel(Tx)	DDR returns data.
N+1	Receive data from bus, write into L2 and L1 ICache. Set state to SHARED.		

Table 7.16: I-Stream Read to an Non Resident Block: With Victim Writeback

Cycle	PX Action	COH Action	DMA/PCI Action	Comment
1	CMD(RDS,COHn,Tx,A,W)			
2		TAG_Check(A) - no hit found. WBC_Check(A) - no hit found. Ty = ORC_Check(A) - HIT! Send A to DDR controller Av = TAG_Victim(A, W)		This can only happen if a block is uncached and then fetched by the DMA/PCI widget via a BRD operation.
3		Shoot down address A in DDR. ORC_Reg(PX, A, Tx) ORC_Dep(Ty, PX, A, Tx) TAG_Update(PX, A, W, SH)		Reads are serviced "in order" even when it "doesn't matter."
L		DATA(PY, Ty, D) – OR	CMD(PRBDONE, COHn, Ty, addr=0)	Data is returned by DDR or DMA/PCI completes a probe to get the data. Either way, COH finds out about it.
L+1		PX, tx, A = ORC_Rel(Ty)		
L+2		Send A to DDR controller		Note contention between this source of addresses and the incoming cmd/addr stream.
N		DATA(PX,Tx,D) ORC_Rel(Tx)		DDR returns data.
N+1	Receive data from bus, write into L2 and L1 ICache. Set state to SHARED.			

Table 7.17: I-Stream Read to a Non Resident Block – Hit on Oustanding Read CAM.

Cycle	PX Action	COH Action	DMA/PCI Action	Comment
1	CMD(RDS,COHn,Tx,A,W)			
2		TAG_Check(A) - no hit found. Ty = WBC_Check(A) - HIT! Ty = ORC_Check(A) - no hit. Send A to DDR controller Av = TAG_Victim(A, W)		We're queued up behind a BWT or a victim writeback.
3		Shoot down address A in DDR. ORC_Reg(PX, A, Tx) WBC_Dep(Ty, PX, A, Tx) TAG_Update(PX, A, W, SH)		
L			DATA(PY, Ty, Dw)	Data is returned by DDR or DMA/PCI completes a probe to get the data. Either way, COH finds out about it.
L+1		PX,Tx, A = WBC_Rel(Ty)		
L+2		Send A to DDR controller		Note contention between this source of addresses and the incoming cmd/addr stream.
N		DATA(PX,Tx,D) ORC_Rel(Tx)		DDR returns data.
N+1	Receive data from bus, write into L2 and L1 ICache. Set state to SHARED.			

Table 7.18: I-Stream Read to a Non Resident Block – Hit on Write Back CAM.

7.10.4.4 I-stream Read to a Cached Block

If an I-stream miss finds the object L2 block in the EXCLUSIVE state, we face something of a problem. If the block is DIRTY, then we need to write the bits in the block back to memory before changing the state of the block to SHARED. (If we don't write the bits to DRAM, and the only copies of this dirty data are in the SHARED state, then the bits may be lost. SHARED blocks can be evicted without being written back.) So, we need to ensure two things. First, that the current owner flushes any dirty data in the block out to main memory. Second, that A eventually arrives at the requesting processor. We do this with a special writeback operation. When PY flushes its data to the coherence widget, the COH will look up the write address, as it always does, in the ORC and WBC. It will find a hit in the ORC. Normally writes don't hit in the ORC, as there are ownership issues at stake here. This write, however, looks like a block write to a cache block that is owned exclusively (except that the current "owner" hasn't seen the data yet.) So we'll leverage the machinery we have sitting around for block writes from cacheless widgets, as described in Sections 7.10.4.8 and 7.10.4.9.

Cycle	PX Action	COH Action	PY Action	Comment
1	CMD(RDS,COHn,Tx,A,W)			
2		TAG_Check(A) - Find at least one match, pick PY. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller Av = TAG_Victim(A, W)		If there is more than one hit in the L2 master tags, then all blocks should be in the SHARED state.
3		CMD(PRBSHR, PY, Tx, A, ORIGIN=Px) Shoot down read of A in DDR controller. ORC_Reg(PX,A,Tx) TAG_Update(PX, A, W, SH)		Send a probe/intervention to PY, asking for block A to be stored in the SHARED state.
L			TAG_Check(A) - If no hit, see Table 7.25.	If A does hit in PY's L2, the state should be SHARED. If not, see Table 7.22.
L+1			DATA(PX,Tx,D)	Send data to processor X
L+2	Receive data from the bus, write to L2 and L1 ICache. Set state to SHARED.			
L+3	CMD(PRBDONE,COHn,Tx, addr=0)			
L+4		ORC_Rel(Tx)		

Table 7.19: I-Stream Read to a Cached Block in SHARED State

Cycle	PX Action	COH Action	PY Action	Comment
1	CMD(RDSV,COHn,Tx,A,W)			
2		TAG_Check(A) - Find at least one match, pick PY. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller Av = TAG_Victim(A, W)		If there is more than one hit in the L2 master tags, then all blocks should be in the SHARED state.
3		CMD(PRBSHR, PY, Tx, A, ORIGIN=Px) Shoot down read of A in DDR controller. ORC_Reg(PX, A, Tx) WBC_Reg(PX, Av, Tx) TAG_Update(PX, A, W, SH)		Send a probe/intervention to PY, asking for block A to be stored in the SHARED state.
M	DATA(COHn,Tx,Dw) or CMD(WBCANCEL,Tx)			This is the victim writeback. M may occur as early as cycle 3.
M+1		Aw = WBC_GetAddr(Tx) send Aw along with the data Dw to the DDR controller. WBC_Rel(PX,Av,Tx)		
L			Lookup A in L2 tags. If no hit, see Table 7.25.	If A does hit in PY's L2, the state should be SHARED. If not, we've got a problem.
L+1			DATA(PX,Tx,D)	Send data to processor X
L+2	Receive data from the bus, write to L2 and L1 ICache. Set state to SHARED.			
L+3	CMD(PRBDONE,COHn,Tx,addr=0)			
L+4		ORC_Rel(PX,A,Tx) from ORC.		

Table 7.20: I-Stream Read to a Cached Block: With Victim Writeback

Cycle	PX Action	COH Action	PZ and PY Action	Comment
1	CMD(RDS,COHn,Tx,A,W)			
2		TAG_Check(A) - Find at least one match, pick PY. WBC_Check(A) - no hit found. Tz = ORC_Check(A) - HIT! Send A to DDR controller		If there is more than one hit in the L2 master tags, then all blocks should be in the SHARED state.
3		Shoot down read of A in DDR controller. ORC_Reg(PX, A, Tx) ORC_Dep(Tz, Px, A, Tx) TAG_Update(PX, A, W, SH)		Register dependency of Tx on Tz.
L		DATA(Pz, Tz, D) or	PZ: CMD(PRBDONE, COHn, Tz, addr=0)	One way or the other, Tz completes – either by getting data directly from the DDR or forwarded from somebody.
L+1		Px, Tx, A = ORC_Rel(Tz)		
L+2		CMD(PRBSHR, PY, Tx, A, ORIGIN=Px)		Send probe command to PY.
M			PY: DATA(Px, Tx, D)	PY Returns data to PX.
M+1	Receive data from the bus, write to L2 and L1 ICache. Set state to SHARED.			
M+2	CMD(PRBDONE,COHn,Tx,addr=0)			
M+3		ORC_Rel(PX,A,Tx)		All done.

Table 7.21: I-Stream Read to a SHARED Block – ORC Hit

Cycle	PX Action	COH Action	PY Action	Comment
1	CMD(RDS,COHn,Tx,A,W)			
2		TAG_Check(A) - Find exactly one match for PY in EXCLUSIVE state. Save matching way in Wy. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller.		This can happen after a I-stream page has been written by the OS or a virus. It would be humiliating to get the wrong answer while executing a virus.
3		CMD(PRBSHR, PY, Tx, A, ORIGIN=Px) Shoot down read of A in DDR controller. ORC_Reg(PX, A, Tx) TAG_Update(PX, A, W, **SH**) TAG_Update(PY, A, Wy, **SH**)		Send a probe/intervention to PY, asking it to invalidate the block. PY will see that the block is EXCL, flush its writes, and will send the data to the COH even if it is clean. Both PY and PX will keep the block in SHARED state.
L			Lookup A in L2 tags. If no hit, see Table 7.25. Probe the L1 blocks and commit L1 updates to the L2 copy.	If A does hit in PY's L2, the state should be EXCLUSIVE. If not, we've got a problem.
L+1			CMD(WRSTRANS, COHn, Ty, Ay, Origin=Tx) Set the state of the L2 copy to SHARED.	Send a writeback and transfer command to COH. Note that we write the data to memory whether it is clean or dirty. It just isn't worth optimizing for this case.
L+2		Px, Tx, A= ORC_CheckS(Tx)		Find the "first" outstanding ORC entry – that's the one that we need to chain on this WRSTRANS
L+3		WBC_Reg(Py, Ay, Ty) WBC_Dep(Ty, Px, A, Tx, **RDS**)		
W			DATA(COH,Ty,Dw)	Send data to the coherence widget. Could occur in the same cycle as L+3.
W+1		Data arrives at COH. Send Dw with Ay to DDR controller. WBC_Rel(Ty)		(This is what we'd do for a RAW hazard. See Section 7.10.4.17.)
M		DATA(Px,Tx,Dw) ORC_Rel(PX, A, Tx)		Coherence controller forwards read data from DDR.
M+1	Receive data from the bus, write to L2 and L1 ICache. Set state to SHARED.			

Table 7.22: I-Stream Read to a Cached Block In EXCLUSIVE State

Cycle	PX Action	COH Action	PY Action	Comment
1	CMD(RDSV,COHn,Tx,A,W)			
2		TAG_Check(A) - Find exactly one match for PY in EXCLUSIVE state, way Wy. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller Send Av (address of victimized block) to WBC.		This can happen after a I-stream page has been written by the OS or a virus. It would be humiliating to get the wrong answer while executing a virus.
3		CMD(PRBSHR, PY, Tx, A, ORIGIN=Px) Shoot down read of A in DDR controller. ORC_Reg(PX, A, Tx) WBC_Reg(PX, Av, Tx) TAG_Update(PX, A, W, SH) TAG_Update(PY, A, Wy, SH)		Send a probe/intervention to PY, asking for block A to be stored in the SHARED state and for PY to EVICT the block.
N	DATA(COHn,Tx,Dv) or CMD(WBCANCEL,Tx)			Cycle M may occur as early as cycle 3. This activity may run in parallel with other parts of the transaction
N+1		Av = WBC_GetAddr(Tx). Send Av along to the DDR controller (along with the data) WBC_Rel(PX, Av, Tx)		

Table 7.23: I-Stream Read to a Cached Block In EXCLUSIVE State: With Victim Writeback

Cycle	PX Action	COH Action	PY Action	Comment
L			Lookup A in L2 tags. If no hit, see Table 7.25. Probe the L1 blocks and commit L1 updates to the L2 copy.	If A does hit in PY's L2, the state should be EXCLUSIVE. If not, we've got a problem.
L+1			CMD(WRSTRANS, COHn, Ty, Ay, Origin=Tx) Set the state of the L2 copy to SHARED.	Send a writeback and transfer command to COH. Note that we write the data to memory whether it is clean or dirty. It just isn't worth optimizing for this case.
L+2		ORC_CheckS(Tx) to find Ax, Tx. Forward this indication to WBC.		In this case OTC_CheckS(A) will match against the first OTC entry that was independent of any other OTC or WBC entry.
L+3		WBC_Reg(PY, Ay, Ty) WBC_Dep(Ty, PX, Ax, Tx, RD)		
W			DATA(COH,Ty,Dw)	Send data to the coherence widget. This could be as early as L+2.
W+1		Data arrives at COH. Send Dw with Ay to DDR controller. WBC_Rel(Ty)		Enqueue Read operation for A,Tx to DDR controller. (This is what we'd do for a RAW hazard. See Section 7.10.4.17.)
M		DATA(Px,Tx,Dw) ORC_Rel(PX, A, Tx)		Coherence controller forwards read data from DDR.
M+1	Receive data from the bus, write to L2 and L1 ICache. Set state to SHARED.			

Table 7.24: I-Stream Read to a Cached Block In EXCLUSIVE State: With Victim Writeback (Continued from Table 7.23)

Cycle	PX Action	COH Action	PY Action	Comment
L			Lookup A in L2 tags and find NOHIT.	This is the continuation of the operations in Tables 7.22 and 7.23.
L+1			CMD(PRBNOHIT,PX,Tx, addr=0)	Send a nohit notification back to processor X. Address and Way are irrelevant.
L+2	Process PRBNOHIT, lookup Tx and find target address and way. Retry the read operation.			
L+3	CMD(RDSR,COHn,Tx,A,W)			
L+4		This is a retry, don't do any tag matching or ORC/WBC lookups, as we don't really care. (And we've already got an ORC registered for this read.) Send A on to DDR controller.		
W		Read data arrives at COH from DDR. ORC_Rel(PX,A,Tx) DATA(Px,Tx,D)		
W+1	Receive data from the bus, write to L2 and L1 ICache. Set state to SHARED.			

Table 7.25: Forwarded I-Stream Read to a Cached Block Misses in Probed Cache

Cycle	PX Action	COH Action	PY Action	Comment
1	CMD(RDS,COHn,Tx,A,W)			
2		TAG_Check(A) - Find exactly one match for PY in EXCLUSIVE state. Save matching way in Wy. WBC_Check(A) - no hit found. Py, Ty = ORC_Check(A) HIT. Send A to DDR controller.		This can happen after a I-stream page has been written by the OS or a virus. It would be humiliating to get the wrong answer while executing a virus. We got an ORC hit because the EXCLUSIVE owner hasn't yet received the block. We need to delay sending the PRB until the block arrives.
3		Shoot down read of A in DDR controller. ORC_Reg(PX, A, Tx) ORC_Dep(Ty, Px, A, Tx, RDS) TAG_Update(PX, A, W, SH) TAG_Update(PY, A, Wy, SH)		Update the blocks to SHARED, that's what they'll be once we're done.
N			CMD(PRBDONE, COHn, Py, Ty, addr=0)	PY finally gets the block it requested.
N+1		Px, A, Tx = ORC_Rel(Ty)		Find the dependent read operation.
N+2		CMD(PRBSHR, PY, A, Tx, ORIGIN=Px)		Ask PY to send data to PX and transition to SHARED.
L	Continue at step L in Table 7.22			

Table 7.26: I-Stream Read to a EXCLUSIVE Block – ORC Hit

Cycle	PX Action	COH Action	DEV Action	Comment
1	CMD(RDS,COHn,Tx,A,W)			
2		TAG_Check(A) - Find exactly one match for PY in EXCLUSIVE state. Save matching way in Wy. ORC_Check(A) - no hit found. DEV, Tv = WBC_Check(A) HIT. Send A to DDR controller.		This can happen after a I-stream page has been written by the OS or a virus. It would be humiliating to get the wrong answer while executing a virus. We got a WBC hit because the EXCLUSIVE block is being updated by a BWT instruction from a DMA/PCI widget.
3		Shoot down read of A in DDR controller. ORC_Reg(PX, A, Tx) WBC_Dep(Tv, Px, A, Tx, RDS) TAG_Update(PX, A, W, SH) TAG_Update(PY, A, Wy, SH)		
N			CMD(BWTDONE, COHn, Ty, addr=0, ORIGIN=Py)	PY finally gets the block it requested.
N+1		Px, A, Tx, Py = WBC_Rel(Tv)		Find the dependent read operation.
N+2		CMD(PRBSHR, PY, Tx, A, ORIGIN=Px)		Ask PY to send data to PX and transition to SHARED.
L	Continue at step L in Table 7.22			

Table 7.27: I-Stream Read to a EXCLUSIVE Block – WBC Hit

7.10.4.5 D-stream Read to a Cached Block in SHARED State

A D-stream read to a block in the SHARED state is a surprise. That is, this is a hint that some process has decided to treat someone's I-stream as data. We need to get this right, but we don't need to make this fast. In any case, this isn't rocket science. The trick here is that we need to make the coherence engine send out invalidates to each of the processors that might have or be acquiring copies of the istream data. Note that victimizing a SHARED block in an L2 does not invalidate the L1 I-cache copy. It is the responsibility of the operating system to see that L1 I-caches remain coherent to the extent it is required. In practical terms, this means that the OS must flush the I-cache when it modifies the I-stream of a process.

Our general approach here is that the COH will send out a broadcast PRBINV command to all caches, directing them to INVALIDATE the target block in their L2 caches. If any processor (other than the requestor) finds the L2 block in the EXCLUSIVE state, we're in trouble and we should signal a machine check.

Cycle	PX Action	COH Action	Other PY Action	Comment
1	CMD(RDEX, COHn, Tx, A, W)			
2		Tag_Check(A) (All matches are in the SHARED state.) WBC_Check(A) – Always misses. ORC_Check(A) – if a hit is found, see Table 7.29. Send address A to DDR controller and queue for DDR read operation.		If one or more of the matching blocks is not in the SHARED state, then we should signal a machine check. Since the block is SHARED, there can be no write transactions outstanding.
3		ORC_Reg(PX, A, Tx) TAG_Update(Px, A, W, EX) For all matching PY: TAG_Update(PY, A, Wy, INV) CMD(PRBINV, BROADCAST, Tx, A)		All PYs are told to invalidate this block in their caches if necessary.
4			Lookup A in L2. Set any matching blocks to “INVALID”. If any matching blocks are EXCLUSIVE, signal a machine check. All processors send CMD(INVDONE, COHx, Tx, A).	INVDONE must be received at the COH from each processor in order to free up the TID Tx.
M		Data Dr returns from DDR. ORC_Rel(Tx)		
M+1		DATA(X, Tx, Dr)		
M+2	Receive data from the bus. Store it in L2 and set state to EXCLUSIVE-CLEAN.			

Table 7.28: D-Stream Read to a Cached Block in SHARED State

Cycle	PX Action	COH Action	Other PY Action	Comment
1	CMD(RDEX, COHn, Tx, A, W)			
2		Tag_Check(A) (All matches are in the SHARED state.) WBC_Check(A) – No hit possible. Py,Ty = ORC_Check(A) Send address A to DDR controller and queue for DDR read operation.		If one or more of the matching blocks is not in the SHARED state, then we should signal a machine check. Note that now there may be two addresses in flight
3		Shootdown address in DDR. ORC_Reg(PX, A, Tx) ORC_Dep(Ty, Px, A, Tx, RDEX) TAG_Update(Px, A, W, EX) For all matching PY: TAG_Update(PY, A, Wy, INV)		All PYs are told to invalidate this block in their caches if necessary.
N		Data returns from DDR OR	PRBDONE arrives from PY	Py's transaction completes
N+1		Px,A,Tx,Opx, Opy = ORC_Rel(Ty)		
N+2		CMD(PRBINV, BROADCAST, Tx, A) Launch address A to DDR		
M		Data returns from DDR ORC_Rel(Tx) DATA(X, Tx, Dr)		Yes, we did two fetches. That's how this works. Otherwise the ORC entries retire out of order.
M+1	Receive data from the bus. Store it in L2 and set state to EXCLUSIVE-CLEAN.		All processors send CMD(INVDONE, COHx, Tx, A).	

Table 7.29: D-Stream Read to a Cached Block in SHARED State ORC Hit

Cycle	PX Action	COH Action	Other PY Action	Comment
1	CMD(RDEX, COHn, Tx, A, W)			
2		TAG_Check(A) (All matches are in the SHARED state.) WBC_Check(A) – nohit. ORC_Check(A) – no hit. Send address A to DDR controller and queue for DDR read operation. Av = TAG_Victim(A, W)		If one or more of the matching blocks is not in the SHARED state, then we should signal a machine check.
3		ORC_Reg(Px, A, Tx) WBC_Reg(PX, Av, Tx) TAG_Update(PX, A, W, EX) For all matching PY: TAG_Update(PY, A, Wy, INV) CMD(PRBINV, BROADCAST, Tx, A)		All PYs are told to invalidate this block in their caches if necessary. BROADCAST is a special Target vector that ensures this command arrives at every port's command queue. (Note that we don't limit the broadcast to processors that have the data.)
4			Lookup A in L2. Set any matching blocks to "INVALID". If any matching blocks are EXCLUSIVE, signal a machine check.	
W	DATA(COHn, Tx, Dw) or CMD(WBCANCEL,Tx)		All processors send CMD(INVDONE, COHx, Tx, A).	W may occur as early as cycle 3.
W+1		Av = WBC_GetAddr(Tx) Send Av along with data Dw to the DDR controller write queue. WBC_Rel(PX, Av, Tx)		
M		Data Dr returns from DDR. ORC_Rel(Tx)		
M+1		DATA(X, Tx, Dr)		
M+2	Receive data from the bus. Store it in L2 and set state to EXCLUSIVE-CLEAN.			

Table 7.30: D-Stream Read to a Cached Block in SHARED State: With Victim Writeback

7.10.4.6 D-Stream Write Miss

D-Stream writes from a processor that miss in its L2 cache require that the L2 segment acquire ownership of the relevant block before the write can complete. Thus, there really isn't a notion of a "D-Stream Write Miss" as L2 write misses become D-Stream Read Miss events described in Sections7.10.4.1, 7.10.4.2, and 7.10.4.5.

7.10.4.7 D-Stream Write to Invalidate

A processor may flush a block from its L2 segment without asking for a refill. In this case, the processor will issue a WINV command as shown in Table 7.31.

If the block to be flushed is clean, then there is no need to send data. In this case, the processor will issue a FLUSH command as shown in Table 7.32. **Note that the FLUSH operation is not implemented in the ICE9 chip. None of the nodes in the chip uses the FLUSH operation.**

Cycle	PX Action	COH Action	Comment
1	CMD(WINV,COHn, Tx,A,W)		W is the way that we'll invalidate. This must match the comparison that will happen for A in the master tags.
2		TAG_Check(A) - Hits on PX.	If there is no tag hit then we've passed a read operation on this block. PX will return a PRBNOHIT to the other ship. So PX needs to write the data back to DDR.
3		TAG_Update(PX, A, W, IN) WBC_Reg(PX, A, Tx)	
M	DATA(COHn,Tx,Dw) – write-back victim block		Cycle M may occur as early as cycle 3. This activity may run in parallel with other parts of this transaction.
M+1		Aw = WBC_GetAddr(Tx) Send Aw along with the data Dw to the DDR controller. WBC_Rel(Tx)	

Table 7.31: D-Stream Write to Invalidate an EXCLUSIVE Dirty Block.

OBSOLETE ——— THIS OPERATION IS NOT IMPLEMENTED IN THE ICE9 V1.0 CHIP ———

OBSOLETE ——— THIS OPERATION IS NOT IMPLEMENTED IN THE ICE9 V1.0 CHIP ———-			
Cycle	PX Action	COH Action	Comment
1	CMD(FLUSH,COHn, Tx,A,W)		W is the way that we'll invalidate. But we already know that from the comparison that will happen for A in the master tags.
2		TAG_Check(A) - Hits on PX.	If there is no tag hit then we've passed a read operation on this block. PX will return a PRBNO-HIT to the other ship.
3		TAG_Update(PX, A, W, IN)	Don't tell anybody else.

Table 7.32: D-Stream Flush to Invalidate and EXCLUSIVE Clean Block.

7.10.4.8 Block Write to a Non Resident Block

As opposed to D-stream Write misses from a processor, I/O and the DMA engine (which we'll also call an I/O device, even though it isn't) may write entire blocks of memory. In this case, we know that all 64 bytes are being written, so there is no need to perform a read of the block and merge in just the changed bytes before the writeback.

On the other hand, we really really want to optimize the path that carries data from a packet buffer in the DMA engine to a processor that will consume it. For that reason, we distinguish block writes that are performed by cacheless device like the DMA engine from those performed by a processor. The "trick" that we're about to employ here would not be appropriate for processors, as the three-stage writeback (a relatively frequent operation) would be a bottleneck for the processors, as they're only allowed one read and one write transaction outstanding at any given time.

So, Tables 7.33 and 7.34 show how a cacheless node on the CSW performs block writes to non resident data.

Cycle	Device Action	COH Action	Comment
1	CMD(BWT,COHn,Tv,A)		Block write from device "DEV"
2		TAG_Check(A) – no hit found. WBC_Check(A) – If there is a hit here, see Table 7.34. ORC_Check(A) – If there is a hit, see Table 7.35.	A hit in the WBC is likely the result of a victimization write from some processor, or – less likely – a colliding write from the DMA engine or the PCI widget. A hit in the ORC is the result of an outstanding BRD.
3		WBC_Reg(DEV, A, Tv) CMD(BWTGO, DEV, Tv, A)	Tell the device to complete its write operation
W	Device receives BWTGO command, matches Tv against outstanding data block to be written.		
W+1	DATA(COHn,Tv,Dw) – send write data block to the coherence widget		
W+2		Receive incoming write data. A = WBC_GetAddr(Tv) Send matching A address to DDR controller along with the data. WBC_Rel(DEV, A, Tv)	

Table 7.33: Block Write to a Non Resident Block

Cycle	Device Action	COH Action	Other Device Action	Comment
1	CMD(BWT, COHn, Tv, A)			
2		TAG_Check(A) – no hit found. Ty, Py = WBC_Check(A) – We find a HIT ORC_Check(A) – There can be no ORC hit.		PY has evicted the block, or another device has launched a write to this block.
3		WBC_Reg(DEV, A, Tv) WBC_Dep(Ty, DEV, A, Tv, BWT)		We'll write the data directly to the DDRAM *after* the victimization write completes.
K			DATA(COH,Ty,Dy)	Other device writes its data to the DDR.
K+1		WBC_Rel(Ty) This causes the dependent write from DEV to be activated. Remove the entry for Ty from the WBC.		
K+2		CMD(BWTGO,Dev,Tv,A)		Contine as at cycle W in Table 7.33.

Table 7.34: Block Write to a Non Resident Block with a Writeback in Flight from Processor Y

Cycle	Device Action	COH Action	Other Device Action	Comment
1	CMD(BWT, COHn, Tv, A)			
2		TAG_Check(A) – no hit found. WBC_Check(A) – no hit. Ty, Py = ORC_Check(A)		BRD outstanding from the other device.
3		WBC_Reg(DEV, A, Tv) ORC_Dep(Ty, DEV, A, Tv, BWT)		We'll write the data directly to the DDRAM *after* the victimization write completes.
K		data returns from DDR OR	CMD(PRBDONE, COHn, Ty, addr=0)	Other device reads its data from the DDR.
K+1		ORC_Rel(Ty) This causes the dependent write from DEV to be activated. Remove the entry for Ty from the ORC.		
K+2		CMD(BWTGO,PX,Tv,A)		Contine as at cycle K in Table 7.36. (If the other device is a processor PY, then we would have seen a TAG_Check hit on PY. We didn't, so we send a BWTGO to PX since the data is not currently cached by anybody.)

Table 7.35: Block Write to a Non Resident Block with a Read in Flight from Processor Y

7.10.4.9 Block Write to a Cached Block

We decided that close integration between the fabric hardware and the processors is really important. We can gain a whole lot of performance over I/O based strategies if we provide a quick path for the DMA engine to return data back to a processor without requiring extra external memory traffic.

For example, consider the "traditional way" that we might implement part of a packet receive operation. You might imagine that the DMA engine would pull the packet off the fabric and write it to DDR memory. Since we have an exclusive/noshare cache coherence protocol, when the DMA engine wrote the data to memory it also invalidated any cached copy of the data. So if processor 0 (P0) does a lot of MPI_RECV operations to the same destination buffer, P0 will have to fetch the received data from memory every time. That could add up to 80nS of overhead for every MPI_RECV operation. But that is the way an I/O based strategy would do this.

On the other hand, the DMA engine is pretty close to the L2 cache segments. So we're not going to invalidate the cached copy of the data unless we have to. In Section 7.33 we described how the DMA engine could do a block write to a non resident block. Table 7.36 shows how this same transaction works when the data is already resident in processor PY's cache. The transaction here assumes that the block is found in the EXCLUSIVE state, and not in the SHARED state. Section 7.10.4.10 describes the transaction flow for the latter operation. Note there are several "bad things" that can happen on the way to completing this operation. In most other transaction tables (above) I've left out the unpleasant paths, deferring discusion until later sections on hazards. I don't do that here, because these hazards are central to the way the transaction works. In particular note, that we never trust to chance in the success of a retry. If a transaction encounters some condition that causes it to restart, we ensure that no other transaction could intervene so as to prevent successful completion. (That's one of the powerful benefits of the chained dependence lists that are maintained in the ORC and WBC structures: once a transaction is registered in the ORC or WBC, it will complete before later dependent operations in the ORC or WBC complete or even attempt to use more L2 switch resources.)

Nonetheless, it is possible that a block write could encounter an ORC hit or WBC hit that causes it to retry, only to find out that the processor holding the block has since evicted it. In this case, the retried operation is garunteed to complete successfully.

All block write transactions carry a "HalfMask" field in the data half of the transaction. This allows the DMA engine to write 32 byte naturally aligned half-blocks to a cached block. HalfMask for a BWT transaction may send 64 bytes, or the first 32 bytes in a block or the last 32 bytes. (See Figures 7.9, 7.10 and 7.11.)

Cycle	Device Action	COH Action	PY Action	Comment
1	CMD(BWT,COHn, Tv,A)			
2		TAG_Check(A) – Find a match against PY, way W. WBC_Check(A) – If there is a hit here, see Table 7.39 and 7.38. ORC_Check(A) – If there is a hit here, see Table 7.40.		A WBC hit implies that there is a colliding victim write or block write in progress. We need to make sure the writes are sequenced in order. An ORC hit implies a read-in-progress and that PY hasn't yet acquired the data, though it has been assigned ownership for block A.
3		CMD(PRBBWT,PY,Tv,A) WBC_Reg(DEV, A, Tv)		
K			PY receives forwarded Block Write command. If A does not hit in the L2, see Table 7.41. Otherwise, invalidate appropriate L1 blocks. Record BWT in progress.	PY could evict a block without informing the COH, or this could be a case of "ships passing in the night."
K+1			CMD(BWTGO,Dev,Tv,A)	

Transaction is continued in Table 7.37.

Table 7.36: Block Write to EXCLUSIVE Cached Data

Cycle	Device Action	COH Action	PY Action	Comment
M	Recieve BWTGO, match Tv against outstanding write.			
M+1	DATA(PY,Tv,Dw) – send write data to processor Y.			
M+2			PY receives data, writes it to L2, removes Tv from list of BWTs in progress	
N			CMD(BWTDONE, COHn, Tv, addr=0)	Note that BWTDONE is sent to coherence engine, not to originating device.
N+1		WBC_Rel(DEV,A,Tv)		There may be dependent writes – see Section 7.10.4.19.

Table 7.37: Block Write to EXCLUSIVE Cached Data (continued from Table 7.36.)

Cycle	Device Action	COH Action	PY Action	Comment
1	CMD(BWT,COHn, Tv,A)			
2		Py, W = TAG_Check(A) – Find a match. Ty, Py = WBC_Check(A) – We find a HIT. ORC_Check(A) – There can be no ORC hit.		A WBC hit implies that there is a colliding victim write or block write in progress. We need to make sure the writes are sequenced in order. The WBC hit could be against a processor's outstanding write, or the PCI widget, or another transaction from this device! In this case, we'll consider writes from PY. For collisions with the PCI or DMA engine, see Table 7.39.
3		WBC_Dep(Ty, DEV, A, Tv, BWT) WBC_Reg(DEV, A, Tv)		PY is recorded as the target, as it matched in the L2 cache lookup.
K			DATA(COH,Ty,Dy)	PY writes its data to the DDR. This is probably a hint that we're going to find that the block has been evicted from the L2 in PY, but we don't know that yet.
K+1		Ay = WBC_GetAddr(Ty) send Ay to DDR WBC_Rel(Ty) – this will wake up Tv.		
K+2		CMD(PRBBWT,PY,Tv,A)		Continue as at cycle K in Table 7.36.

Table 7.38: Block Write to Cached Data – Collision With Outstanding Write from a Processor

Cycle	Device Action	COH Action	Other Device Action	PY Action	Comment
1	CMD(BWT, COHn, Tv, A)				
2		Py, W = TAG_Check(A) – Find a match. Pw, Tw = WBC_Check(A) – Find a hit. ORC_Check(A) – There can be no ORC hit.		case L2_NORD_WT:	A WBC hit implies that there is a colliding victim write or block write in progress. We need to make sure the writes are sequenced in order. The WBC hit could be against a processor's outstanding write, or the PCI widget, or another transaction from this device! In this case, we'll consider writes from the PCI widget or the DMA as the "other device". For a collision with a write from a processor, see Table 7.38.
3		WBC_Dep(Tw, DEV, A, Tv, BWT) WBC_Reg(DEV, A, Tv)			PY's write will wake this write up when it completes.
K			DATA(PY,Tw,Dw)		The writer registered in the WBC completes its write.
K+1				Process incoming data as in Table 7.36.	
M				CMD(BWTDONE, COHn, Tw, addr=0, ORIGIN=OTHER)	This completes the write from the "other" device.
M+1		WBC_Rel(Aw) – We find that Tv is a dependent operation.			
M+2		CMD(PRBBWT,PY,Tv,A)			Contine as at cycle K in Table 7.36.

Table 7.39: Block Write to Cached Data – Collision With Outstanding Write From a Cacheless Widget

Cycle	Device Action	COH Action	PY Action	Comment
1	CMD(BWT,COHn,Tv,A)			
2		TAG_Check(A) – ignore match/nomatch. WBC_Check(A) – no hit. Py, Ty = ORC_Check(A) – hit on access from Py, DMA, or PCI (we'll call it PY for example.)		An ORC hit implies a read-in-progress and that PY hasn't yet acquired the data, though it has been assigned ownership for block A. Since we got a tag match, we should queue up behind the RD transaction, since that's the owner. Otherwise, we should just launch the write, since the read is by a cacheless widget.
3		WBC_Reg(DEV, A, Tv) ORC_Dep(Ty, DEV, A, Tv, BWT)		
K		DDR returns DATA for Ty OR	CMD(PRBDONE, COHn, Ty, addr=0)	PY completes its operation and causes the ORC entry to free up.
K+1		ORC_Rel(PY, A, Ty) which causes the COH to: CMD(BWTGO,Dev,Tv,A)		
M	Recieve BWTGO, match Tv against outstanding write.			
M+1	DATA(PY,Tv,Dw) – send write data to processor Y.			
M+2		WBC_Rel(DEV,A,Tv)		

Table 7.40: Block Write to Cached Data – Collision With Outstanding Read

Cycle	Device Action	COH Action	PY Action	Comment
1	CMD(BWT,COHn, Tv,A)			
2		Py, W = TAG_Check(A) – Find a match. WBC_Check(A) – If there is a hit here, see Table 7.39 ORC_Check(A) – If there is a hit here, see Table 7.40.		A WBC hit implies that there is a colliding victim write or block write in progress. We need to make sure the writes are sequenced in order. An ORC hit implies a read-in-progress and that PY hasn't yet acquired the data, though it has been assigned ownership for block A.
3		CMD(PRBBWT,PY,Tv,A) WBC_Reg(DEV, A, Tv)		
K			PY receives forwarded Block Write command. A does NOT hit in the L2 cache.	PY could evict a block without informing the COH, or this could be a case of "ships passing in the night."
K+1			CMD(BWTNOHIT,Dev,Tv,addr=0)	Tell the device to continue the write to the coherence engine.
M	Recieve BWTNOHIT, match Tv against outstanding write.			
M+1	DATA(COH,Tv,Dw) – send write data to coherence widget, since PY doesn't care.			
M+2		A = WBC_GetAddr(Tv) Send Dw and A to DDR controller for write to DDRAM. WBC_Rel(DEV,A,Tv)		

Table 7.41: Block Write to Cached Data – Encountering an Evicted Block

7.10.4.10 Block Write to SHARED Location

In this case, the block write will invalidate all shared locations and send its data to the DDR controller. The transaction is shown in Table

Cycle	Device Action	COH Action	Comment
1	CMD(BWT,COHn, Tv,A)		Transaction is sent from device "DEV"
2		TAG_Check(A) – Find a match against one or more blocks in the SHARED state. WBC_Check(A) – There can't be a hit in the WBC. WBC_Check(A) – If there is a hit here, see Table 7.43.	A WBC hit implies that there is a colliding victim write or block write in progress. That is inconsistent with the state of the master tags. An ORC hit implies a read-in-progress and that PY hasn't yet acquired the data, though it has been assigned ownership for block A.
3		CMD(PRBINV,BROADCAST,Tv,A) Foreach PY matching in TAG_Check TAG_Update(Py, A, W, INV) WBC_Reg(DEV, A, Tv)	Invalidate all blocks in the SHARED state.
4	Recieve PRBINV, match Tv against outstanding write.		Yup, this is an odd use of PRBINV. But note that any PRBINV that matches the TID for the device's BWT, must be the result of the BWT.
M	DATA(COH,Tv,Dw) – send write data to coherence widget.		All processors send CMD(INVDONE, COHx, Tx, A).
M+1		A = WBC_GetAddr(Tv) Send Dw and A to DDR controller for write to DDRAM. WBC_Rel(DEV,A,Tv)	

Table 7.42: Block Write to SHARED Data

Cycle	Device Action	COH Action	PY Action	Comment
1	CMD(BWT,COHn, Tv,A)			Transaction is sent from device "DEV"
2		TAG_Check(A) – Find a match against one or more blocks in the SHARED state. WBC_Check(A) – No hit. Py, Ty = ORC_Check(A) – Find a hit.		An ORC hit implies a read-in-progress and that PY hasn't yet acquired the data, though it has been assigned ownership for block A.
3		ORC_Dep(Ty, DEV, A, Tv, BWT) Foreach PY matching in TAG_Check TAG_Update(Py, A, W, INV) WBC_Reg(DEV, A, Tv)		We'll activate this BWT when PY completes its read operation. We invalidate the block for the "read in flight" since all future reads will queue up behind our entry in the WBC.
K		DDR returns data for transaction Ty	OR-> CMD(PRBDONE, COHn, Ty, addr=0)	Either COH sees the DDR return data for TID = Ty or PY sends a PRBDONE to the coherence widget.
K+1		ORC_Rel(Ty) See that (DEV, A, Tv) is a dependent operation.		
K+2		CMD(PRBINV, BROADCAST, Tv, A)		
K+3	Recieve PRBINV, match Tv against outstanding write.			
M	DATA(COH,Tv,Dw) – send write data to coherence widget.		All processors send CMD(INVDONE, COHx, Tx, A).	
M+1		A = WBC_GetAddr(Tv) Send Dw and A to DDR controller for write to DDRAM. WBC_Rel(DEV,A,Tv)		

Table 7.43: Block Write to a Cached Block in SHARED State with a Read in Flight from Processor Y

7.10.4.11 Block Write and Other Probe Collisions with Victimization

It is possible that a block write is forwarded to an L2 segment, acknowledged by the segment with a BWTGO command, and then arrives only to find that the target block has been displaced. We could prevent this by locking any block that is the target of a BWTPRB until the data side of the transaction completes. Unfortunately, that smells like a good way to create a deadlock. In fact, this is a problem for probes in general.

The Coherence engine will, of course, detect this when the victimization writeback address matches against the BWT operation in the WBC. But that doesn't help, as the COH has no control over the L2 segment's completion of the victim writeback. The L2 is hell bent for leather on its way to writing the data to DRAM and there's nothing that's going to stop it. (Note that the victimization writeback arrived AFTER the BWT operation was forwarded from the COH, otherwise we'd have held off the continuation of the BWT operation.)

There are lots of ways of handling this, most of them pretty complicated. Since BWT operations are relatively infrequent, and complete quickly, this is what we'll do: (Note that this approach applies to all PROBE operations directed at a processor segment.)

The L2 segment will hold off **all** L1 to L2 read transactions from the processor once it starts processing any kind of probe operation from the CSW. Since only a read operation can cause a victimization, and processors don't execute WINVs, this ensures that a WINV or RDV/RDSV (writeback or victimization of a block) that is initiated before the segment begins processing a PRBBWT (or PRBBRD, PRBSHR, PRBWIN, or PRBINV) has completed before the decision is made to send BWTGO or BWTNOHIT. Further, no new L1 to L2 read transactions are permitted until either a BWTNOHIT, PRBNOHIT, BWTDONE, or PRBDONE has been sent. Fore more detail, see the state machine descriptions of probe handling in the processor segment Section 6.22.

THIS TABLE HAS BEEN REMOVED.

Table 7.44: Block Write Collides with Victimization of Target Block

7.10.4.12 Block Read to a Non Resident Block

The DMA engine or PCI widget will read blocks from memory. This looks much like an RDEX operation in the case of an L2 miss. See Table 7.45.

Cycle	DMA Action	COH Action	Comment
1	CMD(BRD,COHn,Tx,A,W)		
2		TAG_Check(A) - no hit. (or shared) WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller	If there is a WBC hit, see Table 7.47. If there is an ORC hit, see Table 7.46.
3		ORC_Reg(DEV, Tx, A)	
L		DATA(DMA,Tx,D)	Return data from DDR to requester.
L+1		ORC_Rel(Tx)	Note that the ORC wakeup will forward any request to PY rather than the DMA widget, since the DMA has no cache.

Table 7.45: Block Read to Non Resident or SHARED Block

Cycle	DMA Action	COH Action	Comment
1	CMD(BRD,COHn,Tx,A,W)		
2		TAG_Check(A) - no hit. (or shared) WBC_Check(A) - no hit found. Tv = ORC_Check(A) - HIT. Send A to DDR controller	This is sequencing against an RDS or another BRD from device Pv – otherwise we'd be cached EXCLUSIVE.
3		Shootdown A in DDR controller. ORC_Reg(DEV, Tx, A) ORC_Dep(Tv, DEV, Tx, A)	
N		DATA(Pv,Tv,D)	Return data from DDR to original BRD requester.
N+1		DEV, Tx, A = ORC_Rel(Tv)	DEV is DMA
N+2		Send A to DDR	Continue at step L in Table 7.45.

Table 7.46: Block Read to Non Resident or SHARED Block – ORC Hit

Cycle	DMA Action	COH Action	Pv Action	Comment
1	CMD(BRD,COHn,Tx,A,W)			
2		TAG_Check(A) - no hit. Tv = WBC_Check(A) - HIT. ORC_Check(A) - no hit. Send A to DDR controller		This is sequencing against a victim writeback or a BWT from device Pv.
3		Shootdown A in DDR controller. ORC_Reg(DEV, Tx, A) WBC_Dep(Tv, DEV, Tx, A)		
N			DATA(COHn, Tv, D)	Data from writer to DRAM.
N+1		DEV, Tx, A = WBC_Rel(Tv)		
N+2		Send A to DDR		Continue at step L in Table 7.45.

Table 7.47: Block Read to Non Resident or SHARED Block – WBC Hit

7.10.4.13 Block Read to a Cached Block

If the DMA or PCI widget reads a block that is currently in an L2 cache entry, we'll leave it in the L2 cache. The processor segment that currently owns the block will flush its L1 updates (if necessary) to the L2 block and send a copy of the block to the DMA/PCI widget. The state of the cache block will not change. Table 7.48 describes the operation when the read completes after being forwarded to the owner. Table 7.51 shows the sequence when the block is no longer valid by the time the forwarded request arrives.

Cycle	DMA Action	COH Action	PY Action	Comment
1	CMD(BRD,COHn,Tx,A,W)			
2		TAG_Check(A) - Hit on PY in EXCLUSIVE state. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller		If the block is SHARED, see Table 7.45. If there is a WBC hit, see Table 7.49. If there is an ORC hit, see Table 7.50.
3		CMD(PRBBRD,PY,Tx,A) Shoot down read of A in DDR controller. ORC_Reg(DMA, A, Tx)		Send a probe/intervention to PY, asking for block A to be forwarded to DMA.
L			TAG_Check(A) - If no hit, see Table 7.51. Flush L1 dirty to L2 block.	If A does hit in PY's L2, the state should be EXCLUSIVE. If not, we've got a problem.
L+1			DATA(PX,Tx,D)	Send data to processor X
L+2	Accept data.			
L+3	CMD(PRBDONE,COHn,Tx,addr=0)			
L+4		ORC_Rel(Tx)		Note that the ORC wakeup will forward any request to PY rather than the DMA widget, since the DMA has no cache.

Table 7.48: Block Read to Cached EXCLUSIVE Block

Cycle	DMA Action	COH Action	PY Action	Comment
1	CMD(BRD,COHn,Tx,A,W)			
2		TAG_Check(A) - Hit on PY in EXCLUSIVE state. Tv = WBC_Check(A) - HIT! ORC_Check(A) - no hit found. Send A to DDR controller		If the block is SHARED, see Table 7.45.
3		Shoot down read of A in DDR controller. ORC_Reg(DMA, A, Tx) WBC_Dep(Tv, DEV, Tx, A)		Wait on completion of write from Pv.
L			DATA(Pv, Tv, D) OR CMD(BWTDONE, COHn, DEV, Tv, addr=0)	Either way, the write completes.
L+1		DMA, Tx, A, Py = WBC_Rel(Tv)		
L+2		CMD(PRBBRD, PY, Tx, A)		Continue with step L in Table 7.48.

Table 7.49: Block Read to Cached EXCLUSIVE Block – WBC Hit

Cycle	DMA Action	COH Action	PY Action	Comment
1	CMD(BRD,COHn,Tx,A,W)			
2		TAG_Check(A) - Hit on PY in EXCLUSIVE state. WBC_Check(A) - no hit. Tv = ORC_Check(A) - HIT! Send A to DDR controller		If the block is SHARED, see Table 7.45.
3		Shoot down read of A in DDR controller. ORC_Reg(DMA, A, Tx) ORC_Dep(Tv, DEV, Tx, A)		Wait on completion of write from Pv.
L		DATA(Pv, Tv, D) OR	CMD(PRBDONE, COHn, DEV, Tv, addr=0)	Either way, the write completes.
L+1		DMA, Tx, A, Py = ORC_Rel(Tv)		
L+2		CMD(PRBBRD, PY, Tx, A)		Continue with step L in Table 7.48.

Table 7.50: Block Read to Cached EXCLUSIVE Block – ORC Hit

Cycle	DMA Action	COH Action	PY Action	Comment
1	CMD(BRD,COHn,Tx,A)			
2		TAG_Check(A) - Hit on PY in EXCLUSIVE state. WBC_Check(A) - no hit found. ORC_Check(A) - no hit found. Send A to DDR controller Av = TAG_Victim(A, W)		If the block is SHARED, see Table 7.45.
3		CMD(PRBBRD,PY,Tx,A) Shoot down read of A in DDR controller. ORC_Reg(DMA, A, Tx)		Send a probe/intervention to PY, asking for block A to be forwarded to DMA.
L			TAG_Check(A) – no hit.	
L+1			CMD(PRBNOHIT, DEV, Tx, addr=0)	Send a NOHIT to the DMA/PCI.
M	CMD(BRDR, COHn, Tx, A)			
M+1		Ignore tag comparisons and all CAM ops. Send A to DDR controller.		
N		Data arrives from DDR. ORC_Rel(Tx) DATA(DMA, Tx, D)		
N+1	Receive data from the bus and eat it.			

Table 7.51: Block Read to Formerly Cached Block

7.10.4.14 Read from an I/O Location

This is pretty much what you think it might be.[1] Assume for instance that processor X wants to read register R on processor segment Y. Table 7.52 shows the transactino flow.

Cycle	Requester Action	Target Device Action	Comment
1	CMD(RDIO,DEV,Tx,A)		Processor (or PCI/DMA) sends an IO read request to DEV.
2		Match A against registers for this node. Fetch register data.	
N		DATA(X, Tx, D)	Send data back to requestor. Note that this is just one 64 bit word. All the other 7 doublewords in this transfer are set to zero.
N+1	Capture incoming data.		

Table 7.52: I/O Register Read

[1] Surprise!

7.10.4.15 Write to an I/O Location

It turns out that this is more interesting than you might imagine. For a variety of reasons, we've decided that data will never arrive at a processor port unless it has been requested by the processor. [2]So, a write of an I/O register in a processor segment requires that we ask the processor segment to READ some data and load it into the target register!

For example, let's say that processor X wants to write data value D to register R in processor Y. Table 7.53 shows how this will happen.

Cycle	Requester Action	Target Device Action	Comment
1	CMD(WTIO, DEV, Tx, A) Write DATA and BYTEMASK to WTIOREG.		Processor (or PCI/DMA) sends an IO write request to the target device, DEV.
2		Enqueue a RDIO request for the WTIOREG for node X.	
N		CMD(RDIO, X, Tx, WTIOREG) Store A in the WTIOADDR register for this node.	Note that a node can have just one oustanding RDIO or WTIO transaction at a time, so we don't need a stack here.
N+1	DATA(DEV, Tx, DATA, BYTEMASK)		
N+2		Receive DATA and BYTEMASK. Apply both to write the target stored in WTIOADDR register.	

Table 7.53: I/O Register Write

[2](This avoids a whole lot of queueing and buffering and flow-control/backpressure machinery that we could probably get right, but only with more effort than it would be worth.)

7.10.4.16 Read after Read Hazard

Imagine the sequence Read(X,A) followed by Read(Y,A). In this case processor Y's request should be forwarded to X so that we "do the right thing" relative to block ownership and the state of the block.

That's why we have the "Outstanding Read CAM" or ORC. The ORC is indexed by an address or a TID. Each entry contains the TID of a subordinate read and the low bits of the subordinate read address. It is important to note that a transaction will not hit on ORC entry if some previous transaction has already hit on that entry. (This allows us to build a "linked list" of subordinate operations on the ORC. The WBC works in the same way.)

When Read(X,A) arrives it is sent directly to the DDR controller. If A matches a tag in the master tags we shoot the transaction down in the DDR controller. (See section 7.10.4.2.) If A matches a tag in the ORC, we shoot it down in the DDR controller.

In each case, we allocate an entry K in the ORC (it is large enough to accomodate all 14 possible outstanding read operations) for Read(X,A) and record the address and TID.

When Read(Y,A) arrives, we find that it hits in the ORC against entry K. Again we allocate an entry for Read(Y,A) (call it J) and write the TID for Read(Y,A) and low bits of the address into entry K. We also shoot down the Read(Y,A) operation in the DDR controller.

When the DDR controller returns the data for Read(X,A) it also returns the TID for that operation. This TID will hit on entry K. We then read the TID for Read(Y,A) and the low bits of the address from entry K. This is packaged up into an appropriate PROBE request which the coherence controller sends to processor X. When X send the response data to Y, Y will send a PROBE DONE command back to the coherence controller. This will hit against entry J which will then cause a further probe to be sent out if some other processor Z has subsequently hit on entry J with a dependent read.

In any case, the arrival of a ProbeDone or returned read data from the DDR will cause the appropriate entry in the ORC to be marked invalid.

Isn't that among the slicker things that you've seen? Tables 7.54, 7.55, 7.56, and 7.57 show what happens when the ORC entry is released for a cached read operation that has completed.

Unfortunately, read-after-read hazards where the DMA engine or the PCI widget originates the *first* of the two reads (the read that is depended upon) is a little bit stickier. The BRD operation implies that the data is headed for a non-cached user. So we can't send the PRBWIN or the PRBSHR to the DMA/PCI widget the way we did with reads that depended on other processor reads. There are a whole bunch of cases to consider.

Cycle	COH Action	Comment
1	Px, A, Op = ORC_Rel(Ty)	COH completes operation for PY and finds dependent operation (RDEX,RDV) for device PX.
2	CMD(PRBWIN, PY, Tx, A)	Tell PY to give up the block. Continue at Table 7.10 or Table 7.12 at step L

Table 7.54: Read After Read Hazard ORC Release for RDEX, or RDV following RDEX, or RDV

Cycle	COH Action	Comment
1	Px, A, Op = ORC_Rel(Ty)	COH completes operation for PY and finds dependent operation (RDEX,RDV) for device PX.
2	CMD(PRBINV, BROAD-CAST, Tx, A) Send A to DDR Controller.	Tell PY to give up the block. Continue at Table 7.28 at step M.

Table 7.55: Read After Read Hazard ORC Release for RDEX, or RDV following RDS, or RDSV

Cycle	COH Action	Comment
1	Px, A, Op = ORC_Rel(Ty)	COH completes operation for PY and finds dependent operation (RDS,RDSV) for device PX.
2	CMD(PRBSHR, PY, Tx, A, ORIGIN=Px)	Tell PY to give up the block. Continue at Table 7.22 at step L

Table 7.56: Read After Read Hazard ORC Release for RDS, or RDSV following RDEX, RDV, RDS, or RDSV

Cycle	COH Action	Comment
1	DEV, A, Op = ORC_Rel(Ty)	COH completes operation for PY and finds dependent operation BRD for device DEV.
2	CMD(PRBBRD, PY, Tx, A)	Tell PY to supply the block. Continue at Table 7.48 at step L

Table 7.57: Read After Read Hazard ORC Release for BRD following RDEX, RDV, RDS, or RDSV

Cycle	COH Action	Comment
1	Px, A, Tx, Op, Owner, State = ORC_Rel(Ty)	COH completes operation for DMA/PCI and finds dependent operation (RDS,RDSV) for device PX. ORC lookup returns current block owner and current state. In this case, there is no owner.
2	Send A, Tx, Px to DDR controller.	Queue up a DDR transaction on behalf of Px. Continue at step N in the normal flow for the dependent operation on a non-cached block.

Table 7.58: Read After Read Hazard ORC Release for RDEX, RDV, RDS, or RDSV following BRD to an UNCACHED Block

Cycle	COH Action	Comment
1	Px, A, Tx, Op, Owner, State = ORC_Rel(Ty)	COH completes operation for DMA/PCI and finds dependent operation (RDEX,RDV) for device PX. ORC lookup returns current block owner and current state. The current state is EX, the owner is Py.
2	CMD(PRBWIN, Owner, Tx, A)	Continue operation as at step L in Table 7.10.

Table 7.59: Read After Read Hazard ORC Release for RDEX, or RDV following BRD to an EXCLUSIVE Block

Cycle	COH Action	Comment
1	Px, A, Op, Owner, State = ORC_Rel(Ty)	COH completes operation for DMA/PCI and finds dependent operation (RDS,RDSV) for device PX. ORC lookup returns current block owner and current state. In this case, there is no owner. The current state is EX, the owner is Py.
2	CMD(PRBSHR, Owner, Tx, A, ORIGIN=Px)	Continue operation as at step L in Table 7.22.

Table 7.60: Read After Read Hazard ORC Release for RDS, or RDSV following BRD to an EXCLUSIVE Block

Cycle	COH Action	Comment
1	Px, A, Tx, Op, Owner, State = ORC_Rel(Ty)	COH completes operation for DMA/PCI and finds dependent operation (RDS,RDSV) for device PX. ORC lookup returns current block owner and current state. The current state is EX, Py is chosen to respond.
2	CMD(PRBINV, BROADCAST, Tx, A) Send Px, A, Tx to DDR controller.	Continue operation as at step M in Table 7.28.

Table 7.61: Read After Read Hazard ORC Release for RDEX, or RDV following BRD to an SHARED Block

Cycle	COH Action	Comment
1	Px, A, Op, Owner, State = ORC_Rel(Ty)	COH completes operation for DMA/PCI and finds dependent operation (RDS,RDSV) for device PX. ORC lookup returns current block owner and current state. In this case, there is no owner. The current state is SH, Py is chosen to respond.
2	CMD(PRBSHR, Owner, Tx, A, ORIGIN=Px)	Continue operation as at step L in Table 7.22.

Table 7.62: Read After Read Hazard ORC Release for RDS, or RDSV following BRD to an SHARED Block

7.10.4.17 Read after Write Hazard

It is possible that processor X will attempt to read block A just as it is in the process of being evicted from processor Y. There are three possible alignment cases.

First, Read(X,A) arrives at the coherence controller BEFORE Write(Y,A) has arrived. In this case the READ will find that the block is OWNED by Y and the coherence widget will send a PROBE request to Y. Y will complete the write operation with WriteData(Y,A,D). Y will then respond to the probe request that was forwarded on behalf of X with a PRBNOHIT to X. X will re-issue the Read(X,A) command which will arrive at the coherence controller as a read against a block that is non resident. [3]

In the second case, Write(Y,A) arrives, followed by Read(X,A), followed by WriteData(Y,A,D). This is what the WBC (WriteBackCAM) is for. When the COH receives Write(Y,A) it registers the write in the WBC and sets the L2 tag for processor Y to INVALID. WBC is indexed by the address, A and a TID field. Each entry in the WBC contains the address of the write command, the TID for the write command (which is the alternate key), a valid bit, a dependent read TID, and the low bits of a dependent read address. (We need to account for the fact that the address A in Read(X,A) may not be the same as A in Write(A), but refers to the same cache block.) When Read(X,A) arrives, the A matches the address tag in the WBC entry. The TID for Read(X,A) and the low bits of A are recorded in the entry. At the same time, the coherence controller has already sent A on to the DDR controller. The match against an outstanding write causes the COH to send a Read-after-write shootdown signal to the DDR controller to clobber the read in progress. Later on, when WriteData(Y,A,D) arrives, the TID from this transaction will be matched against the secondary key in the WBC. The WBC will send the ADDRESS for the write operation on to the DDR controller (so it will know where to write this incoming data) and sends the read address for Read(X,A) and the TID to the RaW queue in the address path. This read operation is later sent on to the DDR controller when time permits. The key here is that the read operation will arrive at the DDR controller AFTER the write data.

In the last case, Write(Y,A) and WriteData(Y,A,D) both arrive before Read(X,A). In this case, the Read will be processed as a normal read against non resident data. When WriteData(Y,A,D) arrives, the valid bit for the matching entry in the WBC is cleared.

Finally, note that if a Read(X,A) matches an address in the WBC, but the entry already has a recorded dependent read operation, then we consider that the access has MISSED in the WBC. In fact, the operation should have HIT in the ORC since the presence of a read operation in the dependent read field of a WBC entry implies that the read operation has not yet completed.

Cycle	COH Action	Comment
1	Px, A, Op = WBC_Rel(Ty)	COH Completes writeback operation for Py and finds dependent RDEX, RDV, RDS, or RDSV operation for Px.
2	Send A, Tx, Px to DDR controller	Queue up a DDR transaction for Px. Continue at step N in the normal flow for the dependent operation.

Table 7.63: Read After Write Hazard WBC Release for BRD, RDEX, RDV, RDS, or RDSV following BWT, WINV, RDV, or RDSV

[3]Note that we don't forward data from processor Y to X in this case, as the logic and sequencing to avoid the many race opportunities isn't really worth the bother, given this particular sequence should not occur often. Otherwise we need to add extra address comparator machinery in the writeback buffer and all kinds of other junk. The heck with it.

7.10.4.18 Write After Read Hazards

Imagine the sequence Read(X,A), Write(Y,A)... Which version of the data should X receive? The answer is, that it doesn't matter. The only case that matters is Read(X,A), Write(Y,A), Read(Z,A). In this case Z can see the same data as X (both see old data, both see new data) or Z can see newer data than X. But time must not apparently flow backward. We easily handle this as all DDR read transactions to the same bank are processed in order. Further, we know that the WBC will ensure that Read(Z,A) happens AFTER WriteData(Y,A). We also know that Read(X,A) arrived before Read(Z,A) and that Read(X,A) will be processed before Read(Z,A) because of the ordering rules in the address datapath. (Incoming commands on the address path allways take priority over entries in the RaW queue.)

Because of our EXCLUSIVE ownership protocol, there really are only a few opportunities for a WAR hazard.

First, Read(X,A) arrives at the COH just before a victimization writeback command from another processor Y. In this case, X's read will be forwarded to Y and will encounter a NOHIT condition, since reads never hit against victimized blocks. (Note that this simplifies things a bit in the L2 segment design.) When X's read encounters the NOHIT, it will be resent to the coherence controller where it will be turned into a DDRAM read. This will either hit in the WBC, in which case the read will be sent to the DDR after the write has completed, or it will miss in the WBC and be sent to the DDR controller and serviced after the write has completed.

In the second case, Read(X,A) arrives at the COH after the victimization writeback (or block write) command from another processor Y but before the data has arrived. That's what the WBC is for. Read(X,A) will hit against processor Y's write back CAM entry and be enqueued. When Y delivers the writeback data, the WBC entry for Y's transaction will be checked and the subordinate read for processor X will be launched.

In the third case, Read(X,A) arrives at the COH just before a block write command. In this case, the COH will not forward the block write command, as it will HIT against X's ORC entry. This case is covered in Table 7.40.

7.10.4.19 Write After Write Hazards

Because of the EXCLUSIVE ownership scheme that we've adopted, write-after-write hazards can only be caused by a block-write followed by another block-write or a processor's eviction of a block.

In the case of a block-write BW2 following a block-write BW1, the COH will register BW2 as dependent on the BW1 (by updating BW1's entry in the WBC) and refrain from sending out the PRBBWT for BW2 until the COH receives a BWTDONE for BW1.

In the case of a block-write BW1 followed by an eviction writeback VIC, the eviction writeback data must be ignored by the COH unless we can be sure that the evicting processor had a chance to "see" the block write data before it evicted the cache block. We know, that because of the list of outstanding BWT operations in each L2 (See Section 7.10.4.11), incoming BWT data will be reflected by a L2 that has evicted the target block and sent back to the appropriate COH. A victimization that occurs before the BWTGO command is sent out will result in a NOHIT condition in the L2 and is described in Table 7.41. A victimization that occurs after the BWTDONE command is sent out will be processed correctly, as there will be no hit in the WBC at the coherence widget. The "ships passing in the night case" where the BWTPROBE arrives after the RDV/RDSV command has been sent from the processor is covered by the CohWbc module that kills the eviction write if it hits against a BWT in progress.

7.10.5 Interrupt Delivery

The ICE9 Chip doesn't have a central interrupt controller. Instead, we deliver interrupt requests via central switch COMMAND cycles. Each interrupting device is responsible for figuring out which processor should field an interrupt request. When a device needs to signal an interrupt to processor X, for example, it will send a INT command with a reason code to X. The reason code is an eight bit number and an index into the processor's interrupt cause register set.

Interrupts from units that cannot issue CSW commands are delivered by the Slow Interrupts mechanism. See the Slow Interrupts registers in this chapter (section 7.18.8), and see the "Interrupts, Again" section of the Processor Segments chapter, (section 6.19.6).

Cycle	Device Action	PX Action	Comment
1	CMD(INT, PX, CswTid::INT, Reason)		Reason<11:0> is driven on the low bits of the Address bus. All other bits are 0. All interrupts use a constant for the TID, "INT" from the CswTid table.
2		PX writes Reason<7:0> to ICR[Reason<11:8>] and asserts interrupt chosen by Reason<11:9>. Both are cleared under processor (software) control. (See Section 6.19.)	

Table 7.64: Interrupt Delivery

7.10.6 Special Communication Commands

Similar to interrupt delivery, we wanted a special way of moving just a few bits from a processor to the DMA engine. The SPCL command handles this case. SPCL commands are single ended writes that carry all information (both the data and where it is supposed to go) in the Address field of the operation. It is up to the receiving node to "do the right thing" with the incoming operand.

SPCL is triggered by a write to an address in the Spcl address range R_Spcl. (See Section7.18.17.) The physical address and the data are combined to produce a single value that is placed on the CSW address bus. Figure 7.14 shows the layout of the SPCL address and the meaning of the individual fields in the physical address. The only supported destination bus stop is the DMA engine.

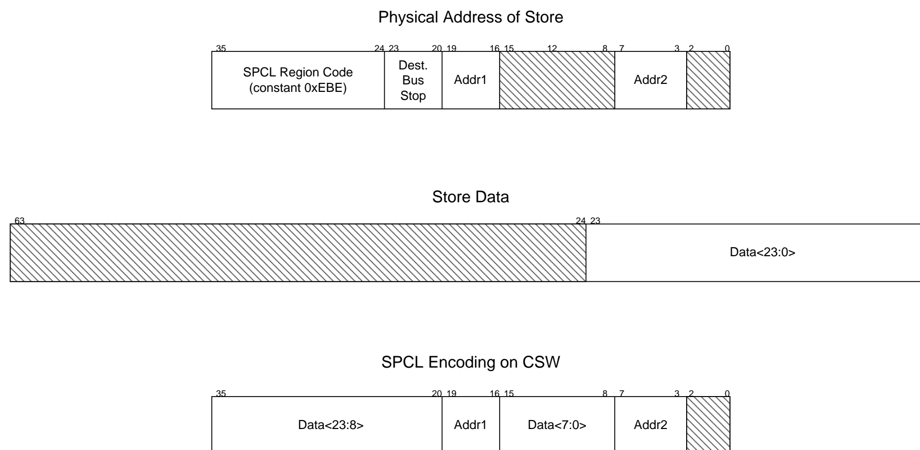


Figure 7.14: SPCL Physical Address Field

Cycle	DMA or Py Action	PX or DMA Action	Comment
1	CMD(SPCL, Px, Td, CmdOp)		CmdOp<35:3> is driven on the Address bus.
2		Px does the right thing with the incoming CmdOp, according to the target module's spec.	
3		CMD(DONE, dev, Td, 0) – tell the sender that the SPCL is done.	

Table 7.65: Special Commands

7.10.7 WINV, Victim Writebacks and the WriteBack CAM

Verification of the protocol encountered this rather gnarly sequence:

1. PS0 does a RDEX for 0x200
2. Sometime later DMA does a BWT to 0x200
3. Before the BWT is complete, PS0 does a WINV where 0x200 is the victim writeback address
4. PS2 does a RDEX for 0x200

So what should happen? Clearly we want the BWT write data to end up in PS2's cache. Were it not for the intervening victim write in step 3 the WBC ordering machinery would just make this happen.

But there are two problems. First, the way the protocol is written the WINV will be entered into the WBC *after* the BWT operation. In fact, it will be registered as a dependent of the BWT. But we know that such writebacks never stall – they're on the way to the DDR and there's nothing that we want to do to stop that. (PS0 will send a BWTNOHIT to the DMA engine after the writeback is complete.) So two bad things will happen: First the WINV will complete and trigger PS2's RDEX. But that will happen before the DMA engines BWT is completed. Second, when the BWT does complete, it will finds its dependent is a WINV or writeback. How do you restart a WINV?

A similar problem happens with RDV and RDSV operations, but in this case the BWT has no dependent registered to it. That is a further problem, since now we'll have TWO entries in the WBC that match the same address and who both think they're the "last" such entry.

We solve this problem with a couple of rules governing WBC_lookup, WBC_dep, and WBC_reg.

(We'll use the phrase "victim writeback" to mean a WINV or the writeback portion of a RDV or RDSV.)

WBC_reg registers a writer. All WINVs, RDVs, and RDSVs, register their write addresses in the WBC. This is as it always has been.

WBC_dep never records WINVs, or the victim writeback portion of RDVs and RDSVs as dependent on a previous entry in the WBC. (There is nothing to wake up.) RDVs and RDSVs may be registered as dependent operations based on the READ addresses.

WBC_lookup may encounter a case where an incoming request matches TWO entries that claim to be "last" in the WBC. In this case, if one of the two entries is a victim writeback, then we pick the other entry as the "parent" in the dependence chain. If neither of the two entries is a victim writeback, then we've got a machine check condition.

Finally (for now) consider the following sequence:

1. PS0 does a RDEX for 0x200 and the read completes
2. PS1 does a RDEX for 0x200
3. PS0 does a WINV (or RDV/RDSV with 0x200 as the victim) before PS1's probe arrives
4. PS2 does a RDEX for 0x200

In this case, the PS2 access will hit in the ORC entry for PS1's RDEX and it will hit on the WBC entry for PS0's victim writeback.

We know that the PS1's RDEX should complete before continuing PS2's RDEX. This is achieved by chaining PS2's access to PS1's access. But what of the victim writeback from PS0?

The answer is that, even though WBC_lookup and ORC_lookup both returned hits for PS2's RDEX, that operation should only be registered as a dependent on the ORC entry, not the WBC entry. In handling all cases where ORC_Hit and WBC_Hit are both true, we behave as if WBC_Hit was false. This works because PS1's RDEX will arrive at PS0 and get a PRBNOHIT *after* PS0 has completed the victim writeback. (Note that this is a requirement on PS0's behavior.)

If not for the intervening read from PS1, PS2's transaction would have missed in the ORC and hit in the WBC. In this case, it would be chained on PS0's WINV completion.

7.11 WRSTRANS and When Bad Things Happen to Good Blocks

WRSTRANS is used to force a transition from some D-stream readable state (EXCL, DIRTY, UPDATED) to SHARED. It comes into play when one processor segment **X** issues an RDS to address A when A is *owned* and in one of the D-stream readable states in some other processor segment **Y**. **X** will send the RDS to COHx

(either COHE or COHO) which will detect the hit on **Y**'s cache and forward a PRBSHR to **Y**. **Y** will then send a WRSTRANS to COHx along with the data from block A. (This is because we're going to give A to **X** which will never write the block back to the DDR, so we have to do the writeback now in case the data is dirty.)

For a whole lot of reasons, we don't have **Y** send data to **X** directly. Instead, there is machinery in the COH that

1. Remembers the last target address for any RDS or RDSV in an array that is indexed by transaction ID.
2. Matches the address of an incoming WRSTRANS (which needs to use a TID from segment **Y** rather than the original TID from **X**'s RDS request. Otherwise the writeback caused by the WRSTRANS could be confused with a writeback from **X** caused by an RDSV.) When an incoming WRSTRANS address matches an entry in the array, COHx will re-issue the read from **X** to the DDR so as to complete the transaction.
3. The DDR read won't be restarted until we're sure the data has been written to the DDR.

7.12 One Thousand Ships, One Thousand Nights

There are a bazillion possible interactions between probes from other processors/devices and outbound requests from a processor segment. Most have tickled one bug or another in the L2 controller or the Coherence widget. Here are a few of them:

7.12.1 Read Retry vs. Victim Writebacks

Imagine that CORE1 sends a PRBWIN A (via the COH) to CORE0 sometime after CORE0 has victimized block A. There are two things to note here: first, CORE0 may respond with a NOHIT before its write data has arrived at the DDR controller; second, the DDR controller does not preserve ordering of read and write commands that arrive from the COH. It is the responsibility of the COH to ensure that no read is issued to the DDR until after any previous writes to that location have made it to the DRAMs.

The ordering is maintained by a mechanism in the COH. When a retry read (RDEXR or RDSR) arrives at the COH, the coh builds a list of all currently outstanding L2 writeback transactions. (That is, all transactions caused by RDSV, RDV, WINV, but not BWT.) If the list is empty, the read retry is sent all the way to the DDR controller without delay. If the list is not empty, the read retry request is queued until each of the transactions in the list have been retired by the DDR controller. (The DDR controller indicates that a write has completed by asserting the ddr_coh_WtTIDVal_c5a signal.) Once the list is empty, the retry reads are resubmitted to the DDR controller and will complete.

7.12.2 PRBWIN A followed by RDEX A

This problem was uncovered by the following trace:

```
((TIME 2144) (FROM CORE2) (TO CORE0) (TID PS2T0) (CMD PRB-
WIN) (ADR #x000000038E8D4FC0) (BMASK #x00) (WAY 0))
((TIME 2152) (FROM CORE0) (TO COH0) (TID PS0T0) (CMD RDEX) (ADR #x000000038E8D4FC0) (BMASK #xFF) (WAY
((TIME 2172) (FROM PCI) (TO CORE0) (TID PS0T0) (CMD PRBNO-
HIT) (ADR #x000000038E8D4FC0) (BMASK #xFF) (WAY 0))
((TIME 2184) (FROM CORE0) (TO CORE2) (TID PS2T0) (CMD PRBNO-
HIT) (ADR #x0000000000000000) (BMASK #x00) (WAY 0))
((TIME 2188) (FROM CORE0) (TO COH0) (TID PS0T0) (CMD RDEXR) (ADR #x000000038E8D4FC0) (BMASK #xFF) (WAY
(((TIME 2200)    (FROM COH0) (TO CORE0)  (TID PS0T0) (MOD_STATE CLEAN)   (HM W64)
   (DW0 #x2DB5EB453184B323) (DW1 #x237A9EF2B4695462) (DW2 #xB81F62A3E366D5D1) (DW3 #x7E3C120113FC9101)
(TIME 2208) (FROM COH0) (TO CORE0) (TID PS0T0) (COMPLETE T))
note that this trace is broken in a couple of ways
```

The first part of this sequence could arise if CORE0 displaces adderss A (000000038E8D4FC0) between the time CORE2's RDEX/V arrived at the COH and the time the PRBWIN arrived at CORE0. When CORE0's RDEX arrives at the COH, we know that it will queue up in the ORC against CORE2's forwarded RDEX. Therefore, the entry at time 2172 in this trace is erroneous, as the COH will not forward the RDEX for PS0T0 until PS2T0 has completed.

On the other hand, PS2T0 must complete so CORE0 must send a PRBNOHIT to CORE2. We solve this problem by noting that the tag array is not updated in the L2 until the fill data has been returned, so any probe lookup against A will miss in CORE0's L2. In this case, for example, CORE0 must send a NOHIT *before* it gets its read data. That's what the CacCtl probe control state machine does.

Note there are problems with the stream that I used for illustration: the PCI should never have sent the PRBNOHIT to CORE0 as the arrival of a PRBWIN for the same address from CORE2 indicates that CORE0's RDEX will wait in the ORC until CORE2's read is complete. Also, PRBNOHIT should be sent with and address of 0.

7.12.3 PRBXXX A While A Is Being Evicted

Imagine that CORE0 owns block A and that CORE1 wants it. CORE1 (via the COH) sends a PRBWIN A to CORE0 just after CORE0 has sent an RDSV B where A is the victim block.

By our normal rule, CORE0 will perform a tag lookup on A and find a HIT. (Note again that L2 tags aren't updated until fill data returns, so the L2 still shows A as valid until B is returned.) But that, of course, is the wrong thing to do. In this case, the CAC must notice that the probe address has hit against a victim writeback and "do the right thing." The actual sequence depends on the type of probe.

7.12.3.1 PRBWIN Against an Evicted Block

Imagine that CORE1 has sent a PRBWIN for A. When we send the PRBNOHIT to CORE1, CORE1 will respond with RDEXR to the COH. The COH read retry handler will "hold" the RDEXR request until all writebacks currently in flight have completed. (See Section 7.12.1.) The only requirement here is that the victim writeback must have been registered in the COH before the RDEXR arrives. This is satisfied if we delay sending the PRBNOHIT until after the data for the victim has been driven onto the CSW. PRBNOHIT responses are delayed in the CMX until any outstanding writeback transactions have completed.

7.12.3.2 PRBSHR Against an Evicted Block

In this case, CORE1 has sent a PRBSHR A to CORE0 which is victimizing A. (Well, at least this isn't going to be as ugly as a WRSTRANS sequence.) CORE0 must hold off sending the PRBNOHIT signal until the outstanding victim write data has been sent. PRBNOHIT responses are delayed in the CMX until any outstanding writeback transactions have completed.

7.12.3.3 PRBBWT Against an Evicted Block

This one appears in BugZilla 860. Imagine the following sequence:

- DMA DMAWT0 BWT A
- CORE0 PS0T1 RDV B, victimize A
- PCI PCIWT0 BWT A

The important thing to ensure is that the writes to memory occur in the following order: CORE0 data, DMA data, PCI data. (Why? Because not all BWT's write all 64 bytes.) How do we do this? Note that the WBC queuing rule will ensure that PCIWT0 is registered as a dependent on DMAWT0. (Not on PS0T1, since a transaction arriving at the WBC will either register as a dependent on a WINV/writeback only if there are no other "last" writers to the target address in the WBC.) So, we need to make sure that DMAWT0 doesn't write its data to the DRAM until after PS0T1's data arrives at the DRAM.

The good news here is that the DDR controller preserves write ordering of blocks with the same address. So, all we have to do is to ensure that CORE0 sends the PRBNOHIT to DMA *after* CORE0 has sent its data to COH. (COH forwards all writeback data along with the writeback address to the DDR when data arrives.) We ensure nohit ordering for PRBBWT responses by requiring that the writebuffer in an L2 segment is empty (or that there are no victim writebacks in progress) before sending a PRBNOHIT. PRBNOHIT responses are delayed in the CMX until any outstanding writeback transactions have completed.

7.12.3.4 PRBBRD Against an Evicted Block

Consider this sequence:

- DMA->COHx DMARD0 BRD A
- CORE0->COHx PS0T1 RDV B, victimize A
- COHx->CORE0 DMARD0 PRBBRD A

In fact, the PRBBRD could come before or just after the RDV. In this case, the appropriate response is NOHIT, but the CORE0 should not send the NOHIT response until it has sent its data back to the COH/DDR. Once the data has been driven onto the CSW, the COH will ensure that the block read retry (BRDR) from DMA will arrive at the COH and queue up until the victim write has made it all the way to the DDR.

7.12.3.5 PRBINV Against an Evicted Block

PRBINV commands require a response. Each processor must return INVDONE to the originating Coherence widget once an PRBINV has been processed. Note that PRBINV commands should only arrive for blocks that are in the SHARED state. The processor never writes back blocks in the shared state, so PRBINV A will never arrive during an eviction of A, though it may arrive while A is being "replaced."

We ran into a nasty protocol issue pretty late in the game. Imagine that thread X is executing Emacs and loads up processor 0's L2 cache with code from Emacs. One of the blocks of Emacs code resides at address A. Now imagine that thread X exits and processor 0 is then used to run a new thread Y. The OS will perform an L1 ICache flush of A, but because we don't communicate cache flush operations to the L2, A still resides in the SHARED state in processor 0's L2. And it contains Emacs code. Imagine that Y is running Quake XVII. Thread Y sends a request to the PCI to page in the code for Quake XVII at location A. The PCI sends a BWT request to the COH which forwards a PRBINV to processor 0. But in this case, processor 0's bus stop is really busy and the PRBINV gets stuck in processor 0's probe queue or even in the incoming command queue in the CSW. Meanwhile the PCI finishes the BWT and sends an interrupt to processor 0. Alas, the interrupt doesn't pass through the probe queue and goes directly to the interrupt register to tell thread Y to go ahead and use the code at block A, as the PCI thinks it is now visible. Thread Y wakes up and executes the OLD instructions in block A (from Emacs) instead of the NEW instructions from Quake XVII. Hilarity ensues.

If we had to do it all over again, we'd probably use some kind of software mechanism, but at this point software invalidates of a page of L2 cache would be very expensive. Instead, we get some help from the protocol.

If a COH sends a PRBINV out in the course of completing a read or write request for TID M from the PCI or DMA, it will set a PRBINV_CTR[M] to 6 and assert TID_BUSY[M] until PRBINV_CTR[M] is zero. (It will also hold TID_BUSY[M] true until the read/write is otherwise complete.) When an INVDONE command arrives with TID = M, PRBINV_CTR[M] will be decremented. Thus, the PCI widget performing a BWT to our address A will not complete the BWT operation until all processors respond with an acknowledgement. (PRBINVs sent out for a TID R belonging to processors (as opposed to the PCI or DMA engine) cause PRBINV_CTR[R] = 5.)

This requires one more adjustment on the part of the PCI (and the DMA engine if it ever overwrites a code page). Interrupts to a processor to signal the completion of a page transfer must not be sent until the last WRITE for that transfer has completed. Further, we may consider whether we want to hold off all PCI writes until the completion (that is, release of the TID) of a BWT that received a PRBINV reply. This ensures that all updates to memory appear in order.

7.12.4 PRBXXX A Just Prior to Evict Attempt on A

The arrival of a probe request at a processor segment may not be processed by the CacCtl unit for several cycles. Thus a probe arriving before an eviction attempt may not be processed until well after the eviction attempt. In that case, we follow the paths described above. If the probe is processed before the eviction attempt, the receiving segment will send a BWTGO or PRBWIN response *before* allowing the processor to initiate the L2 access that would have caused the victimization.

In the case of a PRBBWT arriving just prior to what would have been an eviction attempt, the CAC will hold off all processor accesses to the L2 until the BWT operation has completed.

In the case of a PRBWIN arriving just prior, the CAC will hold off all processor access to the L2 until after the tag array has been updated and the block made invalid.

In the case of a PRBSHR or PRBBRD, the CAC will hold off processor access to the L2 until after data has been read from the L2 and sent to the requesting device.

7.12.5 Implications for Stimulus Generators and Checkers

The sections above describe what the Cac and Coh will do in response to a number of "ships passing in the night" sequences. The responses and rules have some implications for stimulus generators and BFMs.

7.12.5.1 NOHIT sequencing against writeback data

A processor segment (PS) will never emit the following sequence:

- RDV A (victim B)
- PRBNOHIT (for address B)
- WRITE DATA (for address B)

This is impossible since the eviction of B will cause the processor to defer responding with a PRBNOHIT until after the victim data has been sent out onto the CSW.

7.13 Command Fields

Certain of the CSW commands require an address or bytemask or some other value to be meaningful. Other commands stand on their own. Table 7.66 shows the required fields for each of the CSW command types. Where a field is *not* required by a command, it should be driven as 0 and ignored at the receiver.

Command	Address	Way	TID	BMask
IDLE				
RDS, RDSV, RDEX, RDV, RDEXR, RDSR	Y	Y	Y	
WINV	Y		Y	
WBCANCEL			Y	
RDIO	Y		Y	Y
WTIO	Y		Y	Y
BWT, BRD, BRDR	Y		Y	
PRBINV, PRBWIN, PRBSHR, PRBBRD, PRBBWT	Y		Y	
PRBNOHIT, BWTNOHIT			Y	
WRSTRANS	Y	Y	Y	
PRBDONE, BWTDONE			Y	
SPCL	Y		Y	
INT	Y		Y	
BWTGO	Y		Y	

Table 7.66: CSW Commands, Required Fields

7.14 Transaction IDs (TIDs) and TID Busy Signals

Among the CSW signals described in Tables 7.1 and 7.2 are the TIDBusy signals. These are used to indicate to a CSW client that the corresponding TIDs are in flight within either the even or odd coherence controller.

A TID is "in flight" in a coherence widget if

1. The TID corresponds to a valid entry in the ORC, or
2. the TID corresponds to a valid entry in the WBC, or
3. the TID was attached to a read operation sent to the DDR that has not yet either returned data or been shot down, or
4. the TID was attached to a write operation sent to the DDR that has not yet "completed" in the eyes of the DDR controller.

Each coherence widget originates its own version of the TID busy wires. At each bus stop, the TIDBusy output is the result of ORing the TID busy bits from the EVEN COH and from the ODD COH. The COHE TID busy wires are cohe_csw_TIDBusy_c4a[27:0] and the corresponding COHO wires are coho_csw_TIDBusy_c4a[27:0].

The TIDBusy bits from each of the coherence widgets are ORed together and distributed by the CSW after being flopped. For a processor/L2 segment PS0, the CSW output is

```
csw_ps0_TIDBusy_c5a[0] = coho_csw_TIDBusy_c4a[PS0T0] || cohe_csw_TIDBusy_c4a[PS0T0];
csw_ps0_TIDBusy_c5a[1] = coho_csw_TIDBusy_c4a[PS0T1] || cohe_csw_TIDBusy_c4a[PS0T1];
```

For the DMA and PCI widgets, the CSW outputs are

```
csw_dma_RdTIDBusy_c5a[3:0] = coho_csw_TIDBusy_c4a[DMARD3:DMARD0] | cohe_csw_TIDBusy_c4a[DMARD3:DMARD0
csw_dma_WtTIDBusy_c5a[3:0] = coho_csw_TIDBusy_c4a[DMAWT3:DMAWT0] | cohe_csw_TIDBusy_c4a[DMAWT3:DMAWT0
```

Internal to the COH widgets, TIDs are tracked for both WRITE and READ operations. That is, a TID that involves both a read and a write is the logical OR of the RD TID Busy state machine output, the ORC valid bit for this TID, and the WBC valid bit for this TID. The Read TID Busy state machine is described in Figure 7.15.

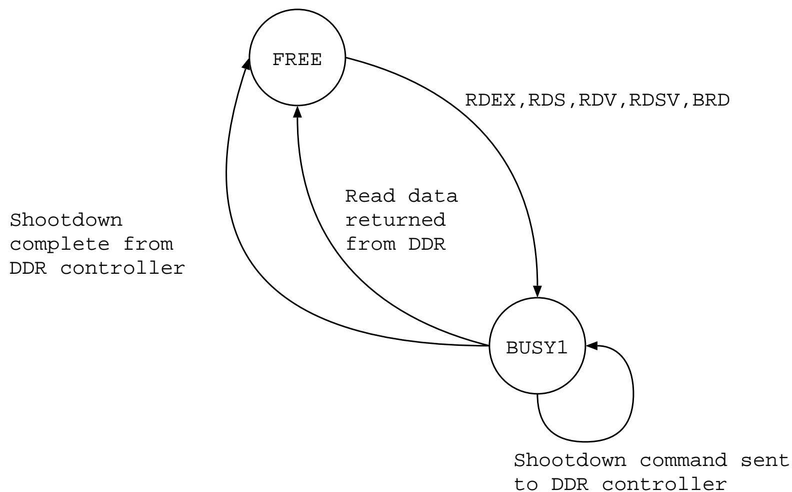


Figure 7.15: Read TID Busy State Machine

The tracking mechanism depends on several signals from the DDR controller

ddr_coh_DataTID_c2a<4:0> The TID for a read data operation that is about to complete

ddr_coh_DataValid_c2a If true, the corresponding TID has sucessfully completed a read of DDR memory. Perform an ORC_Release on the outstanding transaction and cycle the RDTID Busy state machine back to the free state.

ddr_coh_RdShotDown_c2a If true, the corresponding TID's read operation was shot down. Cycle the RDTID busy state machine back to the free state.

ddr_coh_WtTID_c5a<4:0> The TID of a write operation that has passed the ordering point in the DDR controller, that is, we now know that the write for this TID has been sent to the DDR DIMMS when the WtTIDVal bit is set

ddr_coh_WtTIDVal_c5a When true, the corresponding TID should case a WBC_REL operation.

7.14.1 TID Allocation – the IO and MEM TID Spaces

To avoid a nasty and obscure deadlock situation, the processor segment must allow a cache read/replacement operation to proceed in parallel with an IO write or, potentially, an IO read operation. This could require that TID1 for a processor segment (normally used for RDV, RDSV, and IOWT operations) be used by both an IO

write and an RDV/RDSV at the same time. We allow this by treating TIDs belonging to processors as existing in two different spaces. PSnT0 and PSnT1 (where n is in the range 0 to 5) can represent transactions in IO space or memory space. If the accompanying command is an IOWT, IORD, SPCL, INT, or DONE, the TID should be treated as an IO space TID. Otherwise it is for a memory space transaction. If the accompanying data is a double word, then the TID should be treated as an IO space TID. TID busy only reports the condition of TIDS in memory space. IO operations may be emitted from a processor segment if the required TID is not otherwise occupied in IO space.

7.15 The Parts

7.15.1 The Coherence Controller (COH)

7.15.1.1 Block Diagram

The Coherence Controllers (Instance names are COHE for "Even" side coherence widget, and COHO for "Odd" side coherence widget.) field data transfer requests from the six processors, the PCI controller, and the DMA engine. In addition, each coherence controller services I/O requests for the configuration registers in its associated DDR controller.

Each coherence controller contains

- Six 2K by 44 bit TAG arrays (parity protected)
- One 14 entry Outstanding Read CAM that can be indexed by virtual address bits 35:7, or by a six bit transaction ID. Its payload is the Transaction ID and low address bits of the dependent operation, and a Valid bit.
- One 14 entry WriteBack CAM that can be indexed by VA<35:7> or by the TID. Its payload is the TID and low address bits of the dependent operation, and a valid bit.

The CAMs, being implemented in flip-flops, rather than RAM cells, need not be ECC protected. The SER (soft error rate) for the Tag RAMs is such that we'd see a TAG error about once every 30 years. On the other hand, a bit error in the Tags could cause us to generate a "wrong" result or launch the missile, so we'll parity protect the RAMs and force a system recovery if an error is detected.

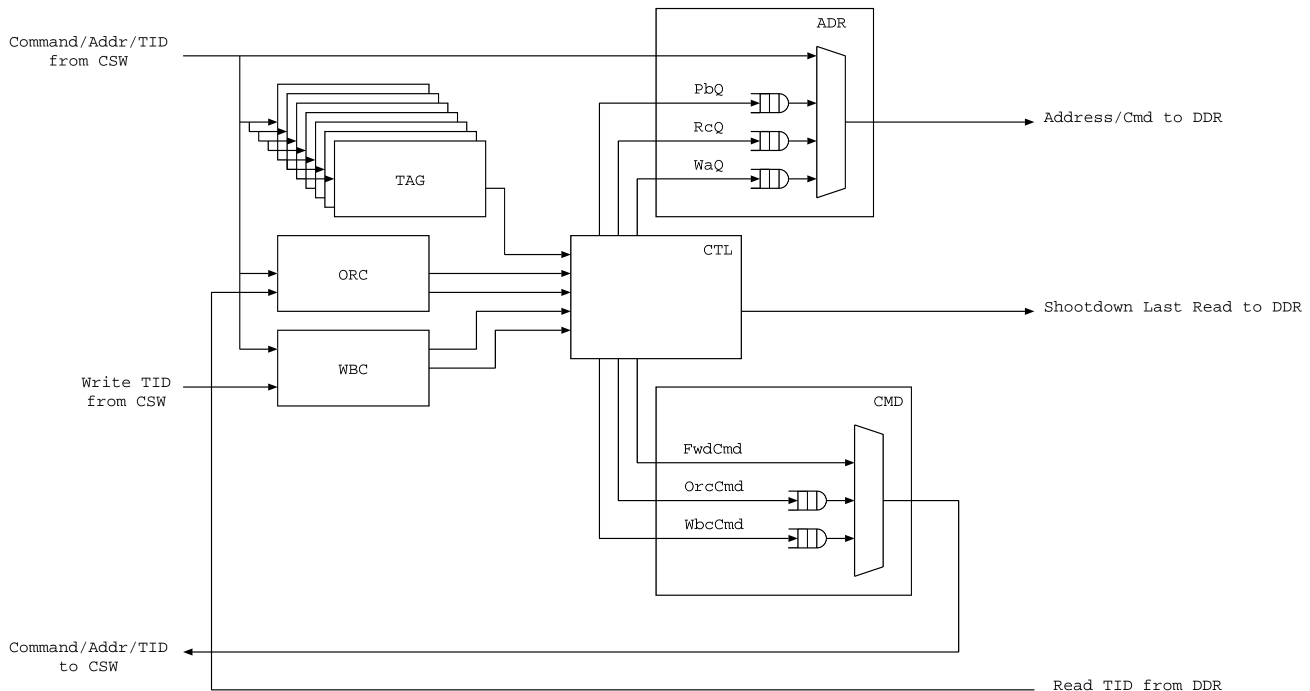


Figure 7.16: Coherence Controller Block Diagram

7.15.1.2 Processing Pipeline(s)

Commands are processed in a two or four stage pipeline that begins in C3 (to align with the pipelines from the processor segments, the PCI and the DMA engine.)

C3: In C3 we look the address up in each of the Master Tags arrays, the ORC, and the WriteBack CAM.

C4: In C4 we update the Tag arrays, the ORC, and the WriteBack CAM. We also send out any transaction operations on the outbound command ports.

Data returns from the DDR controller in C10. (This is an arbitrary choice, but it seems to fit well with the rest of the pipeline definitions in the DMA engine, etc. Data is written into the DDR controller in C3 and following cycles. Figure 7.17 shows the four major processing pipelines in the coherence engine.

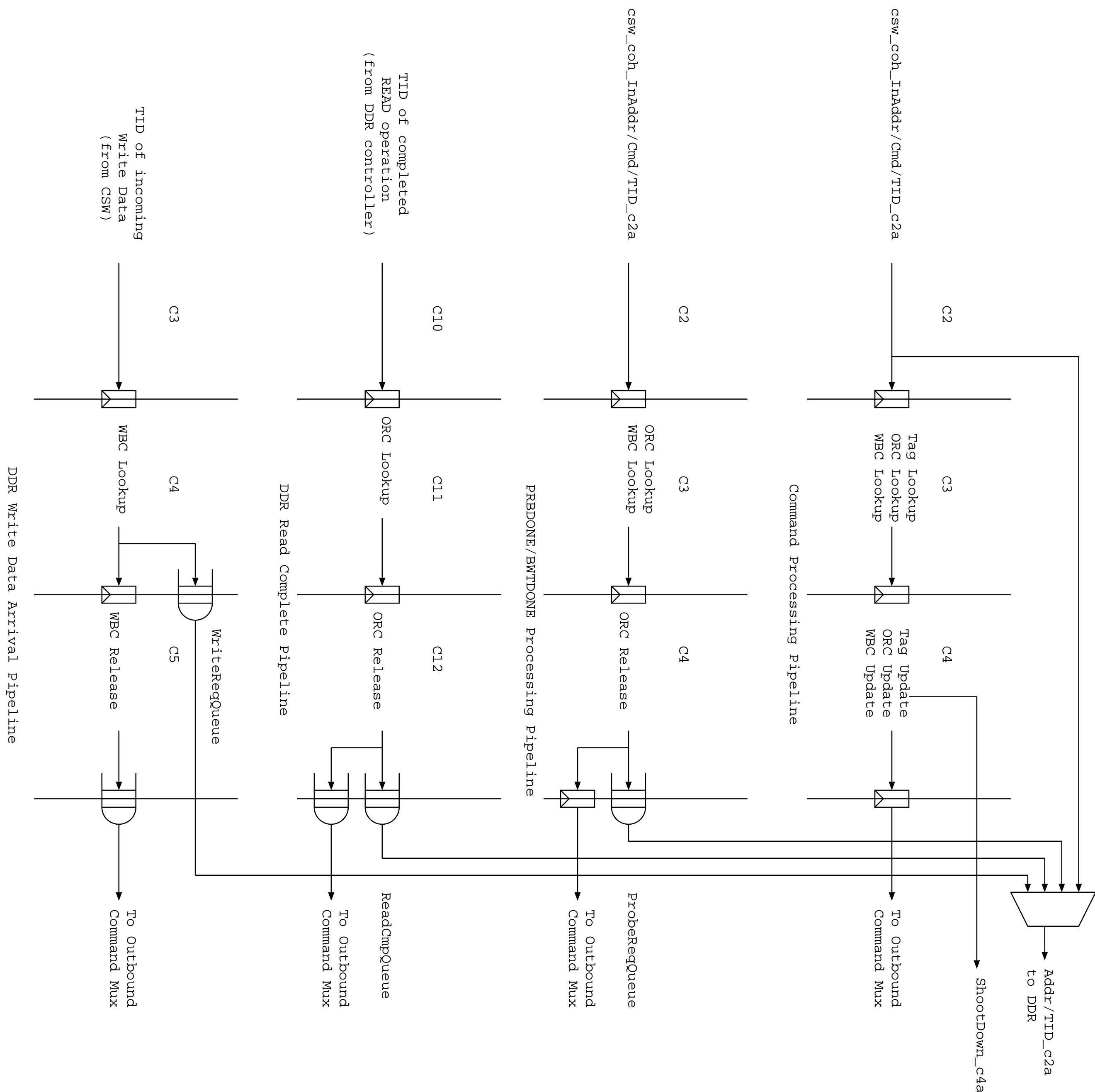


Figure 7.17: Coherence Engine Processing Pipelines

Tables 7.67, 7.68, and 7.69 all assume an incoming command from processor or unit Px. If the target block is owned, Py is the owner.

Normally, an operation can't hit in both the WBC and the ORC in the same cycle. PRBDONE and WRSTRANS are the only exceptions. Ignore the WBC hit in these cases.

Note that WRSTRANS hitting on an EXCLUSIVE block means that we saw a sequence like (RDS,Px,A) (RD,Pz,A) (WRSTRANS,Py,A) where processor Z flipped the L2 cache states from SH to INV in Px before the transaction completed. This is OK. Everything will eventually complete in order and Px will have seen the block for a short time in the SH state before answering a PRBINV broadcast. (This is the reason we write EXCLUSIVE blocks back to DDR rather than sending the data directly from Py to Px.)

7.15.1.3 Recovering from Tag ECC Errors

As it turns out, the master tag arrays contain about 500K bits of storage. We're likely to see a soft error rate on the SRAM cells of about 3000 failures per billion hours per Mbit. So, assuming 1000 chips in a system:

$$MTTF = \frac{10^9\,hours * 1\,Mbit}{1000\,nodes * 3000\,failures * 0.5\,Mbit} = 666\,hours = 28\,days$$

That means that we'd see a tag parity failure about once per month. We can't really recover from that kind of error so we'd have to crash the node and probably the rest of the cluster. That's one of the problems with welding the fabric so close to the processors – if a processor sneezes, the fabric catches pneumonia. So, we need to innoculate the processors by building ECC into the tag RAMs. (Note that we don't need to do this for the CAMs since they're implemented in much more robust flip-flops.)

The tag rams cycle at 4nS, so we have more than enough time to do ECC scrubbing and correction. Tag entries are written in the second stage of the command processing pipeline, so we have enough time to calculate the ECC before the tag update cycle.

7.15.2 The L2 Switch (CSW)

7.15.2.1 Bus Stops, Node Numbers, and Transaction Targets

7.16 Arbitration at the PS to CSW Port

Commands issued by the CAC (RDS, RDSV, RDEX, RDV), the processor (IOWT, IORD, SPCL, INT), or in response to probe operations (WRSTRANS, BWTGO, PRBNOHIT) all must contend at the output of the CAC/PS for the outbound command request wires. Arbitration between these request streams is more complicated than one would hope, but simulation and detailed analysis suggest that the scheme is not prone to deadlock. (Neither simulation nor logical argument can ever garuntee freedome from deadlock, but we do the best we can.) This section describes the arbitration rules and makes the argument that no combination or sequence of requests can cause any one request to remain starved for access to the CSW.

The arbitration is a hybrid priority based and round robin scheme. First, any requests that must be retried from a previous cycle are garunteed access to the outbound command wires. Second, if an L2 cache miss access (RDS, RDSV, RDEX, RDV) was not driven on to the command bus in the previous tic, the waiting request wins. Third, if a NOHIT response causes the CAC to issue a RDSR or RDEXR (read retry), the retry request is driven onto the bus. Fourth if there are no waiting requests, but the L2 tag lookup in the previous cycle has resulted in an L2 miss, the requested command (RDSV, RDV, RDS, RDEX) is driven onto the command wires. If none of these conditions obtains, then we move on to the group of requests that arbitrate in "round robin" fashion. (Note that the priority based portion of the arbitration is deadlock free as the CAC only supports one outstanding memory read access at a time. Thus an RDSR will never contend with an outgoing RDEX, and an L2 miss will never contend with a previously queued memory request.)

Ten different sources of outbound command requests contend in the second stage of arbitration:

LOCINVDONE: Sends out an INVDONE in response to a PRBINV that arrives at this CAC due to a read operation that was initiated by this CAC.

WRSTRANS: Sends out a Write Shared Transition command in response to a PRBSHR on a block held in the exclusive state.

Incoming Command	ORC Miss, WBC Miss	ORC Hit, WBC Miss	WBC Hit, ORC Miss
RDEX	Launch read to DDR. Update L2 Tags for PX. Add PX request to ORC.	Kill read to DDR. Update L2 Tags for PX. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Kill read to DDR. Update L2 Tags for PX. Add PX request to ORC. Add PX dependence on PY transaction in WBC.
RDV	Add Victim Address to WT Queue and WBC. Otherwise, identical to RDEX		
RDS	Launch read to DDR. Update L2 Tags for PX. Add PX request to ORC. (RDSV: Add victim address to WT Q.)	Kill read to DDR. Update L2 Tags for PX. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Kill read to DDR. Update L2 Tags for PX. Add PX request to ORC. Add PX dependence on PY transaction in WBC.
RDSV	Add Victim Address to WT Queue and WBC. Otherwise, identical to RDS		
BRD	Launch read to DDR. Add PX request to ORC.	Kill read to DDR. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Kill read to DDR. Add PX request to ORC. Add PX dependence on PY transaction in WBC.
BWT	Send BWTGO to requester. Add PX request to WBC.	Queue transaction dependence on PY in ORC. Add PX request to WBC.	Queue transaction dependence on PY in WBC. Add PX request to WBC.
RDSR	Retry event reacting to "NOPROBE" response: Launch read to DDR.		
RDEXR	Retry event reacting to "NOPROBE" response: Launch read to DDR.		
WINV	Error! Writeback from non-owning processor! Complete write, update L2 Tags, Declare Machine Check Exception.	WINV from PX passed an inflight PRBWIN for this block. Add Addr to WT Queue and WBC.	WINV from PX passed an inflight BWT for this block. Kill transaction in Write Queue, as the BWT takes precendence.
FLUSH **(UNUSED)**	Error! Flush from non-owning processor! Update L2 Tags (invalidate). Declare Machine Check Exception.		
RDIO	RDIO Transactions Never Arrive at COH		
WTIO	WTIO Transactions Never Arrive at COH		
PRBDONE	Error! Should hit on ORC entry.	Activate matching ORC Entry.	Ignore.
WRSTRANS	Error! WRSTRANS should hit on the L2 Tag for the original requesting processor.		
BWTDONE	Error! Should hit on WBC entry.	Ignore	Activate matching WBC entry.
IDLE	Cancelled operation – do nothing.		

Table 7.67: Coherence Controller Command Pipe Actions vs. Tag and CAM Lookups (For transactions that miss in L2 Master Tags)

Incoming Command	ORC Miss, WBC Miss	ORC Hit, WBC Miss	WBC Hit, ORC Miss
RDEX	Send read to DDR. Update L2 Tags for PX to EX. Invalidate L2 Tags for ALL other matchers. Add PX request to ORC. Broadcast PRBINV to all nodes.	Kill read to DDR. Update L2 Tags for PX to EX. Invalidate L2 Tags for all other matchers. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Not Possible. (If a write is outstanding against the block, why is in SHARED state?)
RDV	Add Victim Address to WT Queue and WBC. Otherwise, identical to RDEX		
RDS	Kill read to DDR. Update L2 Tags for PX to SH. Add PX request to ORC. Send PRBSHR command to “first matcher” PY.	Kill read to DDR. Update L2 Tags for PX to SH. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Not Possible.
RDSV	Add Victim Address to WT Queue and WBC. Otherwise, identical to RDS		
BRD	Kill Read to DDR. Send PRBBRD to PY. Add PX request to ORC.	Kill read to DDR. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Not Possible.
BWT	Send BWTGO to Px. (Note the COH will send PRBINV *after* BWTDONE arrives.) Add PX address to write queue. Add PX request to WBC Note need for PRBINV.	Queue transaction dependence on PY in ORC. Add PX request to WBC.	Not Possible.
RDSR	Retry event reacting to “NOPROBE” response: Launch read to DDR.		Queue dependency on PY in WBC. (We passed an invalidate transaction.)
RDEXR	Retry event reacting to “NOPROBE” response: Launch read to DDR.		Queue dependency on PY in WBC. (We passed an invalidate transaction.)
WINV	Error! WINV should only arrive for exclusively owned blocks unless we have a ships-passing-in-the-night problem (ORC or WBC hit).	Add victim to WT queue and WBC. We’ll wait for the RDSR/RDEXR.	Collision with a BWT. Kill this write when it arrives. Create no WT queue or WBC entry.
FLUSH **(UNUSED)**	Invalidate L2 Tags for PX.		
RDIO	RDIO Transactions Never Arrive at COH		
WTIO	WTIO Transactions Never Arrive at COH		
PRBDONE	Error! Should hit on ORC entry.	Activate matching ORC Entry.	Ignore.
WRSTRANS	Error! WRSTRANS should hit on the ORC entry for the transaction that caused it.	Find FIRST ORC entry for this address (ORC_CheckS). Add Addr to WT Queue and WBC. See Table 7.22 steps L+2 and following. (Note there may be a spurious WBC hit for this operation. Ignore it.)	Ignore.
BWTDONE	Error! Should hit on WBC entry.	Ignore	Activate matching WBC entry. Broadcast PRBINV to all.
IDLE	Cancelled operation – do nothing.		

Table 7.68: Coherence Controller Command Pipe Actions vs. Tag and CAM Lookups (For transactions that *hit* in L2 Master Tags in SHARED State.)

Incoming Command	ORC Miss, WBC Miss	ORC Hit, WBC Miss	WBC Hit, ORC Miss
RDEX	Kill read to DDR. Update L2 Tags for PX. Invalidate L2 Tags for PY. Add PX request to ORC. Send PRBWIN command to PY.	Kill read to DDR. Update L2 Tags for PX. Invalidate L2 Tags for PY. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Kill read to DDR. Update L2 Tags for PX. Invalidate L2 Tags for PY. Add PX request to ORC. Add PX dependence on PY transaction in WBC.
RDV	Add Victim Address to WT Queue and WBC. Otherwise, identical to RDEX		
RDS	Kill read to DDR. Update L2 Tags for PX. Update L2 Tags for PY to SH. Add PX request to ORC. Send PRBSHR command to PY.	Kill read to DDR. Update L2 Tags for PX. Update L2 Tags for PY to SH. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Kill read to DDR. Update L2 Tags for PX. Update L2 Tags for PY to SH. Add PX request to ORC. Add PX dependence on PY transaction in WBC.
RDSV	Add Victim Address to WT Queue and WBC. Otherwise, identical to RDS		
BRD	Kill read to DDR. Send PRBBRD to PY. Add PX request to ORC.	Kill read to DDR. Add PX request to ORC. Add PX dependence on PY transaction in ORC.	Kill read to DDR. Add PX request to ORC. Add PX dependence on PY transaction in WBC.
BWT	Send PRBBWT to PY. Add PX request to WBC.	Queue transaction dependence on PY in ORC. Add PX request to WBC.	Queue transaction dependence on PY in WBC. Add PX request to WBC.
RDSR	Retry event reacting to "NOPROBE" response: Launch read to DDR.		
RDEXR	Retry event reacting to "NOPROBE" response: Launch read to DDR.		
WINV	Add Addr to WT Queue and WBC. Invalidate L2 Tags for PX.	WINV from PX passed an inflight PRBWIN for this block. Invalidate L2 Tags for PX. Add Addr to WT Queue and WBC.	WINV from PX passed an inflight BWT for this block. Invalidate L2 Tags for PX. Kill transaction in Write Queue, as the BWT takes precendence.
FLUSH **(UNUSED)**	Invalidate L2 Tags for PX.		
RDIO	RDIO Transactions Never Arrive at COH		
WTIO	WTIO Transactions Never Arrive at COH		
PRBDONE	Error! Should hit on ORC entry.	Activate matching ORC Entry.	Ignore.
WRSTRANS	Error! WRSTRANS should hit on the ORC entry for the transaction that caused it.	Find FIRST ORC entry for this address (ORC_CheckS). Add Addr to WT Queue and WBC. See Table 7.22 steps L+2 and following. (Note there may be a spurious WBC hit for this operation. Ignore it.)	Ignore.
BWTDONE	Error! Should hit on WBC entry.	Ignore	Activate matching WBC entry.
IDLE	Cancelled operation – do nothing.		

Table 7.69: Coherence Controller Command Pipe Actions vs. Tag and CAM Lookups (For transactions that *hit* in L2 Master Tags in EXCLUSIVE State.)

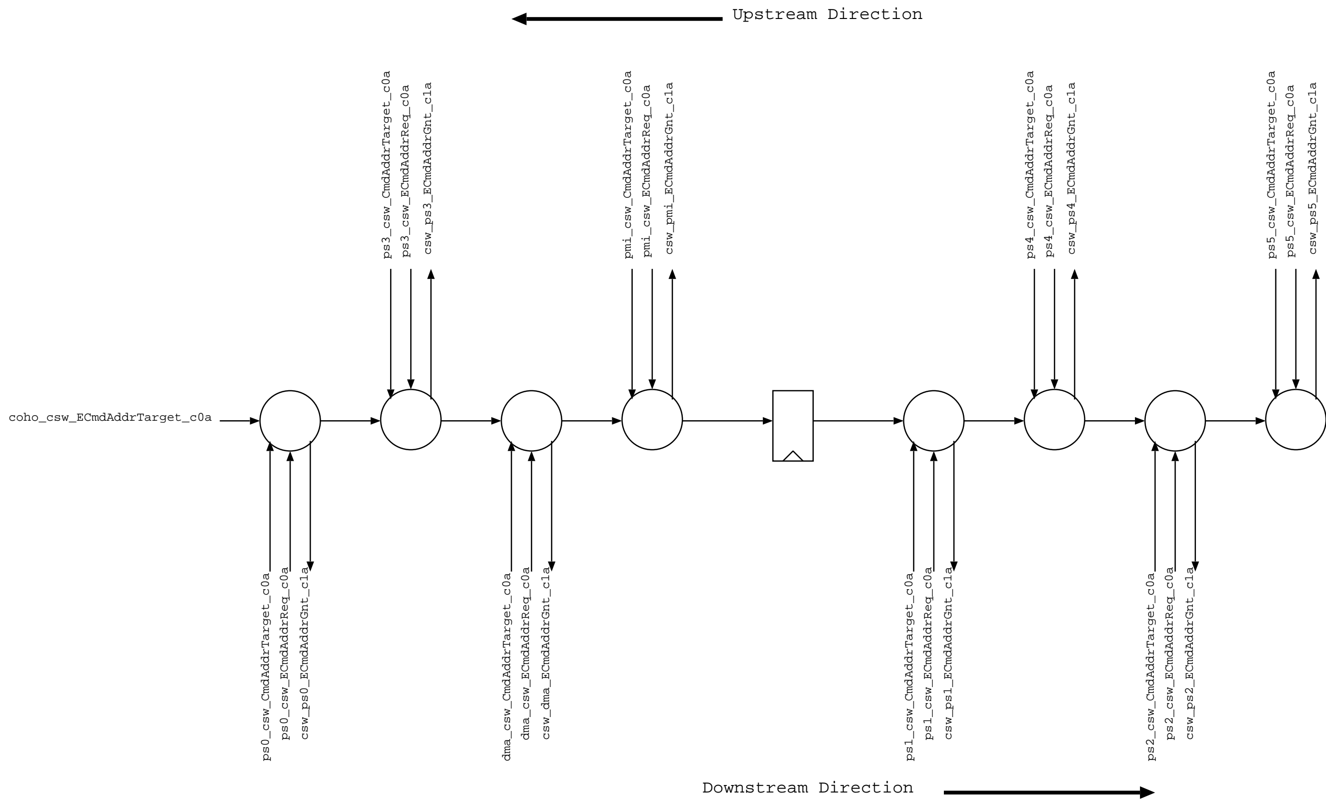


Figure 7.18: Even Bound Command/Address Arbitration Chain

STAGE5: These commands (PRBNOHIT, BWTNOHIT, BWTGO, INVDONE) are in response to probe commands arriving from other nodes.

DONE: These commands (BWTDONE, PRBDONE) are issued in response to completion of a BWT or probe transaction.

INTW: This stream sends out an INT command to deliver an interrupt to another processor segment.

SPCL: This stream sends out a SPCL command to the DMA.

IDONE: This stream sends out a DONE command to signal completion of an INT delivery.

VICCAN: This stream sends out a WBCANCEL command to rescind a writeback request for a block that is now known to be clean.

IORD: This stream sends out RDIO commands.

IOWT: Surprise! This stream sends out WTIO commands.

The arbitration passes through two stages. In stage 1, the ten sources each determine their eligibility to bid. For instance, the IOWT source may not bid for access to the command bus if a previous IO write is in flight, or if the IORD stream has an *earlier* IORD waiting, or if the most recent IO write sent out its data in the last 8 tics or so. In stage 2, all the eligible bidders compete. The highest priority bidder rotates round-robin. The round-robin pointer is bumped each time some stream wins the arbitration. (It is not bumped if there were no requesters, and it is not bumped if a memory read command is being driven because of the priority based arbitration described above.)

So how could we create a starvation case? Assume some request stream A needs resource X to be eligible. Now assume a second stream B also needs X. If A and B both arb at the same time and B wins, A will lose. Now A can't bid again until B releases its resource. If B releases its resource and then needs to rearb again, it may beat B again. In fact, if B releases its resource and *any other stream* requires resource X, A could lose again. Round-robin

arbitration will not prevent this kind of starvation. So, we need to make sure that if A loses a round of arbitration, it will eventually become the only eligible requester that requires its resource. How do we do that?

First, we can dismiss all commands streams that require no resource at all to become eligible. This eliminates the **LOCINVDONE**, **DONE**, **STAGE5**, **IDONE** and **VICCAN** streams.[4]These require no resources, so even in the worst case they only wait for the round-robin pointer to make them the highest priority choice.

Second we should note that the remaining memory related stream **WRSTRANS** only requires a free TID. In this case, the requirement is that either TID0 or TID1 be available for a memory transaction. (If the TID is being used by a RDIO, WTIO, INT, or SPCL, it is still available for use by a memory operation.) Since only one memory transaction can be in flight at a time, and we never need to do two WRSTRANS operations at a time, there is no "B" request stream that could starve out the **WRSTRANS** stream if it was "A" in the above example. Note that nothing that happens with IO related operations ever contends with memory operations. So now, having dismissed arbitration conflicts among memory operations, we only need consider starvation among the IO operations.

First, because of the strict ordering of **IORD** and **IOWT** operations from the core, we never allow an **IOWT** to pass an **IORD** or vice-versa. This means that **IORD** operations never contend with **IOWT**. So they can't starve each other out.

Alas, there has to be a fly in the ointment somewhere. The **IOWT** stream requires the IOWriteTID to be available. So does the **INTW** and **SPCL** stream. This is the lone known opportunity for starvation in the CMX. A pathological program could issue an IOWT request and follow it with a sequence of writes to the interrupt delivery or SPCL delivery register so as to prevent the IOWT from ever completing. However, we require a SYNC either before or after any SPCL or INTW write in order to ensure proper delivery of the SPCL operation. This would prevent the IOWT from starving. In any case, the IOWT was likely not performed from user mode, as we aren't likely to allow user mode programs to fiddle with IO space, even if we do allow (and encourage) usermode access to the SPCL registers. Programs that send out back-to-back SPCLs without SYNCs get what they deserve. (A SYNC instruction would stall the processor and prevent further SPCL writes until the stalled IOWT had completed.)

7.17 Definitions and Enumerations

7.17.1 Package Attributes

Package

chip_cac_spec

7.17.2 Definitions

Defines

CAC

Constant	Mnemonic	Definition
32'h18_0000	SIZE	L2 Cache Size. Total size in bytes including all banks.
32'hC	ASSOC	L2 Cache Associativity.
8'd15	CMD_ADDR_FIFO_DEPTH	Depth of Command/Addr FIFOs for all bus stops
8'd1	DATA_PS_FIFO_DEPTH	The PS only needs one slot on each side
8'd5	DATA_PCI_FIFO_DEPTH	Depth of Data FIFOs for PCI bus stop
8'd3	DATA_DMA_FIFO_DEPTH	Depth of Data FIFOs for DMA bus stop

The PCI DATA fifo depth must be 5 to cover the fact that the PCI widget could have three BRDs, one RDEX, and two WTIO operations completing at one time *and it takes 4 cycles to consume a FIFO entry.* The DMA widget only needs 3 as it can remove an entire entry on every tic, and need only support four BRDs and one WTIO completion. If all five transactions arrived sequentially at a DMA port from the same direction, we'd peel them off in order and only need one slot in the queue to accomodate them. The worst case for DMA is three from the Even side and two from Odd.

[4]A reader of the CacCmxBeh sources will note that STAGE5 requests all require that there be no queued victim writebacks. This condition is redundant, as we already ensure that no writebacks are in progress before issuing the requests from the probe control state machine. A similar condition on **VICCAN** requests is only a delaying mechanism, as we only allow one read miss in flight at a time. It looks redundant now, but we aren't going to remove this logic, as all reasoning is subject to verification, and we've got lots of verification cycles on this logic.

7.17.3 Processor to L2 Cache Commands

This section has been removed.

7.17.4 L2 Cache to Processor Commands

This section has been removed.

7.17.5 L2 Cache to/from Coherence Controller Commands

Enum

CohCmd

Constant	Mnemonic	Definition	(I/O Device Use)
5'b00111	IDLE	Idle	-
5'b00000	RDS	Read shared (instruction)	-
5'b00001	RDSV	Read shared, write victim	-
5'b00100	RDSR	Read shared, retry	-
5'b00010	RDEX	Read data exclusive	Output
5'b00011	RDV	Read data exclusive, write victim	-
5'b00110	RDEXR	Read data exclusive, retry	
5'b00101	RDIO	Read from I/O space	Input
5'b01000	WRSTRANS	Write retaining shared copy	-
5'b01001	WTIO	Write to I/O space	Input
5'b01011	WINV	Writeback and Invalidate	
5'b01010	FLUSH	FLUSH block from L2 cache – no writeback	**NOT IMPLEMENTED**
5'b01110	DONE	WINV, INT, or SPCL is complete	
5'b01111	WBCANCEL	Cancel writeback request from RDSV and RDV	
5'b10001	BWT	Block Write	Input
5'b10000	BRD	Block Read	
5'b10010	BWTNOHIT	Block Write encountered evicted block	
5'b10100	BWTGO	Continue Block Write	
5'b10101	BWTDONE	Block Write Complete	
5'b10110	BRDR	Block Read Retry	
5'b11000	PRBINV	Probe to invalidate	-
5'b11001	PRBWIN	Probe to writeback/transfer	-
5'b11010	PRBSHR	Probe to share block	
5'b11011	PRBBRD	Probe to forward Block Read	
5'b11110	PRBBWT	Probe to forward Block Write	
5'b11100	PRBDONE	Probe transfer completion	Output
5'b11101	PRBNOHIT	Probe finds no block resident	-?
5'b11111	INVDONE	PRBINV acknowledge	
5'b01100	INT	Interrupt request	
5'b01101	SPCL	Special Command	

7.17.6 L2 Cache Coherence Widget States

Enum

CohState

Constant	Mnemonic	Definition
2'b00	INV	Invalid
2'b01	EXCL	Exclusive
2'b10	SHARE	Shared
2'b11	UNUSED	Unused encoding

7.17.7 L2 Segment Cache States

Enum
CacState

Constant	Mnemonic	Definition
3'b000	INV	Invalid
3'b001	EXCL	Exclusive
3'b010	SHARE	Shared
3'b110	DIRTY	Different from Memory Copy
3'b111	UPDATED	Different from Memory and Updated since last fill.

7.17.8 L2 Cache Modified States

Enum
CohModState

Constant	Mnemonic	Definition
2'b00	INV	block is invalid
2'b10	DIRTY	block was modified at some point wrt DRAM copy
2'b11	UPDATED	block was written by the current owner since last ownership transfer
2'b01	CLEAN	block is unmodified wrt DRAM copy

7.17.9 L2 Half Block Update Tags

Enum
CohHalfMask

Constant	Mnemonic	Definition
2'b00	W64	Whole block of 64 bytes
2'b01	L32	Half block of 32 bytes on Dat0..Dat3
2'b10	H32	Half block of 32 bytes on Dat4..Dat7
2'b11	I8	I/O Transaction of just 8 Bytes on Dat0

7.17.10 L2 Cache Interface Numbers (Bus Stop Numbers)

This enumeration contains the physical bus stop number, used to route on the cache switch. For software interrupts, and addressing, the similar AddrStopNum 16.6.5 is used instead. (Thus, this table may change without affecting any software.)

Enum
CswStopNum

Constant	Mnemonic	Definition
4'b0000	COHO	coherence controller on odd side
4'b0110	PCI	PCI controller
4'b0010	CORE0	L2 segment for core 0
4'b0001	CORE1	L2 segment for core 1
4'b0100	CORE2	L2 segment for core 2
4'b0101	CORE3	L2 segment for core 3
4'b1000	CORE4	L2 segment for core 4
4'b0111	CORE5	L2 segment for core 5
4'b0011	DMA	dma controller
4'b1001	COHE	coherence controller on even side
4'b1111	BROADCAST	Broadcast to all nodes (legal from COHE or COHO only

7.17.11 L2 Cache Interface Numbers (Bus Stop Numbers) for TWICE9

This enumeration contains the physical bus stop numbers (for TWICE9), used to route on the cache switch. This new set enumerations has been created because using the old emnumeration would mean that the constant for COHE could not be changed (as this would break for ICE9A) but it also can't be redefined (since enumerations

don't support this). If TWICE9 required keeping the old value for COHE, this would require significant coding changes to verilog and system C code in Cac, Dma, Coh, and PMI).

Enum

CswStopNumTwc

Constant	Mnemonic	(Product)	Definition
4'd0	COHO	TWC9A+	coherence controller on odd side
4'd1	CORE1	TWC9A+	L2 segment for core 1
4'd2	CORE0	TWC9A+	L2 segment for core 0
4'd3	DMA	TWC9A+	dma controller
4'd4	CORE2	TWC9A+	L2 segment for core 2
4'd5	CORE3	TWC9A+	L2 segment for core 3
4'd6	PCI	TWC9A+	PCI controller
4'd7	CORE5	TWC9A+	L2 segment for core 5
4'd8	CORE4	TWC9A+	L2 segment for core 4
4'd9	CORE7	TWC9A+	L2 segment for core 7
4'd10	CORE6	TWC9A+	L2 segment for core 6
4'd11	CORE9	TWC9A+	L2 segment for core 9
4'd12	CORE8	TWC9A+	L2 segment for core 8
4'd13	COHE	TWC9A+	coherence controller on even side
4'd14		TWC9A+	Reserved
4'd15	BROADCAST	TWC9A+	Broadcast to all nodes (legal from COHE or COHO only

7.17.12 Transaction IDs

Enum

CswTid

Constant	**Mnemonic**	Definition
5'd0	PS0T0	Any op for PS0
5'd1	PS0T1	Any op for PS0
5'd2	PS1T0	Any op for PS1
5'd3	PS1T1	Any op for PS1
5'd4	PS2T0	Any op for PS2
5'd5	PS2T1	Any op for PS2
5'd6	PS3T0	Any op for PS3
5'd7	PS3T1	Any op for PS3
5'd8	PS4T0	Any op for PS4
5'd9	PS4T1	Any op for PS4
5'd10	PS5T0	RDE/RDS/FLUSH/RDIO for PS5
5'd11	PS5T1	Any op for PS5
5'd12	DMARD0	BRD 0 for DMA
5'd13	DMAWT0	BWT 0 for DMA
5'd14	DMARD1	BRD 1 for DMA
5'd15	DMAWT1	BWT 1 for DMA
5'd16	DMARD2	BRD 2 for DMA
5'd17	DMAWT2	BWT 2 for DMA
5'd18	DMARD3	BRD 3 for DMA
5'd19	DMAWT3	BWT 3 for DMA
5'd20	PCIRD0	BRD 0 for PCI
5'd21	PCIWT0	BWT 0 for PCI
5'd22	PCIRD1	BRD 1 for PCI
5'd23	PCIWT1	BWT 1 for PCI
5'd24	PCIRD2	BRD 2 for PCI
5'd25	PCIWT2	BWT 2 for PCI
5'd26	PCIRD3	BRD 3 for PCI
5'd27	PCIWT3	BWT 3 for PCI
5'd31	INT	used for all INT commands from all blocks

7.17.13 Transaction IDs for TWICE9

Enum
CswTidTwc

Constant	**Mnemonic**	(Product)	Definition
6'd0	PS0T0	TWC9A+	Any op for PS0
6'd1	PS0T1	TWC9A+	Any op for PS0
6'd2	PS0T2	TWC9A+	Any op for PS0
6'd3	PS0T3	TWC9A+	Any op for PS0
6'd4	PS1T0	TWC9A+	Any op for PS1
6'd5	PS1T1	TWC9A+	Any op for PS1
6'd6	PS1T2	TWC9A+	Any op for PS1
6'd7	PS1T3	TWC9A+	Any op for PS1
6'd8	PS2T0	TWC9A+	Any op for PS2
6'd9	PS2T1	TWC9A+	Any op for PS2
6'd10	PS2T2	TWC9A+	Any op for PS2
6'd11	PS2T3	TWC9A+	Any op for PS2
6'd12	PS3T0	TWC9A+	Any op for PS3
6'd13	PS3T1	TWC9A+	Any op for PS3
6'd14	PS3T2	TWC9A+	Any op for PS3
6'd15	PS3T3	TWC9A+	Any op for PS3
6'd16	PS4T0	TWC9A+	Any op for PS4
6'd17	PS4T1	TWC9A+	Any op for PS4

6'd18	PS4T2	TWC9A+	Any op for PS4
6'd19	PS4T3	TWC9A+	Any op for PS4
6'd20	PS5T0	TWC9A+	Any op for PS5
6'd21	PS5T1	TWC9A+	Any op for PS5
6'd22	PS5T2	TWC9A+	Any op for PS5
6'd23	PS5T3	TWC9A+	Any op for PS5
6'd24	PS6T0	TWC9A+	Any op for PS6
6'd25	PS6T1	TWC9A+	Any op for PS6
6'd26	PS6T2	TWC9A+	Any op for PS6
6'd27	PS6T3	TWC9A+	Any op for PS6
6'd28	PS7T0	TWC9A+	Any op for PS7
6'd29	PS7T1	TWC9A+	Any op for PS7
6'd30	PS7T2	TWC9A+	Any op for PS7
6'd31	PS7T3	TWC9A+	Any op for PS7
6'd32	PS8T0	TWC9A+	Any op for PS8
6'd33	PS8T1	TWC9A+	Any op for PS8
6'd34	PS8T2	TWC9A+	Any op for PS8
6'd35	PS8T3	TWC9A+	Any op for PS8
6'd36	PS9T0	TWC9A+	Any op for PS9
6'd37	PS9T1	TWC9A+	Any op for PS9
6'd38	PS9T2	TWC9A+	Any op for PS9
6'd39	PS9T3	TWC9A+	Any op for PS9
6'd40	DMARD0	TWC9A+	BRD 0 for DMA
6'd41	DMAWT0	TWC9A+	BWT 0 for DMA
6'd42	DMARD1	TWC9A+	BRD 1 for DMA
6'd43	DMAWT1	TWC9A+	BWT 1 for DMA
6'd44	DMARD2	TWC9A+	BRD 2 for DMA
6'd45	DMAWT2	TWC9A+	BWT 2 for DMA
6'd46	DMARD3	TWC9A+	BRD 3 for DMA
6'd47	DMAWT3	TWC9A+	BWT 3 for DMA
6'd48	DMARD4	TWC9A+	BRD 4 for DMA
6'd49	DMAWT4	TWC9A+	BWT 4 for DMA
6'd50	DMARD5	TWC9A+	BRD 5 for DMA
6'd51	DMAWT5	TWC9A+	BWT 5 for DMA
6'd52	DMARD6	TWC9A+	BRD 6 for DMA
6'd53	DMAWT6	TWC9A+	BWT 6 for DMA
6'd54	PCIRD0	TWC9A+	BRD 0 for PCI
6'd55	PCIWT0	TWC9A+	BWT 0 for PCI
6'd56	PCIRD1	TWC9A+	BRD 1 for PCI
6'd57	PCIWT1	TWC9A+	BWT 1 for PCI
6'd58	PCIRD2	TWC9A+	BRD 2 for PCI
6'd59	PCIWT2	TWC9A+	BWT 2 for PCI
6'd60	PCIRD3	TWC9A+	BRD 3 for PCI
6'd61	PCIWT3	TWC9A+	BWT 3 for PCI
6'd62		TWC9A+	Reserved
6'd63	INT	TWC9A+	used for all INT commands from all blocks

7.17.14 Address Tag and Index Fields for L2 and Coh Tag and Data arrays

Defines

CADDR_FLD

Constant	Mnemonic	Definition
64'h040	BANK_SEL_MSK	Which bit selects the "bank" (i.e. EVEN or ODD side COH)
16'd10	HASH_WIDTH	How wide is hashed portion of the tag index?
16'd7	HASHLO_START	Where does the low half of the tag hash field start?
16'd17	HASHHI_START	Where does the hi half of the tag hash field start?
16'd18	TAG_WIDTH	How wide is the stored address tag?

7.17.15 L2 Cache Useful Dimensions

Defines
CAC_DIM

Constant	Mnemonic	Definition
16'd2048	L2TAGARR_SIZE	Number of entries in L2 Tag Array
16'd8192	L2DATWARR_SIZE	Number of Quadwords (16 bytes) in each WAY of the L2 Data Array

7.17.16 Coherence Engine Useful Dimensions

Defines
COH_DIM

Constant	Mnemonic	Definition
4'd14	DCQ_ENTRIES	Number of entries in Data Completion Queue
4'd14	CCQ_ENTRIES	Number of entries in Command Completion Queue
4'd14	PBAQ_ENTRIES	Number of entries in probe completion ORC release address queue
4'd14	RCAQ_ENTRIES	entries in read complete ORC release address queue
4'd14	WCAQ_ENTRIES	entries in write complete WBC release address queue
4'd14	WDAQ_ENTRIES	entries in the write address queue o
4'd14	RCQ_ENTRIES	entries in the ORC dependent command queue
4'd14	WCQ_ENTRIES	entries in the WBC dependent command queue
16'd1024	MTAG_ENTRIES	number of tags per way per L2 Master Tag Array
8'd28	ORC_ENTRIES	number of slots in the oustanding read CAM
8'd28	WBC_ENTRIES	number of slots in the writeback CAM
8'd27	MAX_TID	maximum transaction ID value

7.17.17 Coherence Engine Useful Dimensions for Twice9A

Defines
COH_DIM_TWC

Constant	Mnemonic	(Product)	Definition
6'd52	DCQ_ENTRIES	TWC9A+	Number of entries in Data Completion Queue
6'd52	CCQ_ENTRIES	TWC9A+	Number of entries in Command Completion Queue
6'd52	PBAQ_ENTRIES	TWC9A+	Number of entries in probe completion ORC release address queue
6'd52	RCAQ_ENTRIES	TWC9A+	entries in read complete ORC release address queue
6'd48	WCAQ_ENTRIES	TWC9A+	entries in write complete WBC release address queue
6'd48	WDAQ_ENTRIES	TWC9A+	entries in the write address queue o
6'd52	RCQ_ENTRIES	TWC9A+	entries in the ORC dependent command queue
6'd48	WCQ_ENTRIES	TWC9A+	entries in the WBC dependent command queue
16'd1024	MTAG_ENTRIES	TWC9A+	number of tags per way per L2 Master Tag Array
8'd64	ORC_ENTRIES	TWC9A+	number of slots in the oustanding read CAM
8'd64	WBC_ENTRIES	TWC9A+	number of slots in the writeback CAM
8'd63	MAX_TID	TWC9A+	maximum transaction ID value

7.17.18 Coherence Engine L2 Tag Array Fields

Defines
COH_MTAG

Constant	Mnemonic	Definition
8'd0	TW0_LOW	Low bit of Way 0 Tag
8'd19	TW1_LOW	Low bit of Way 1 Tag
8'd19	TAG_WIDTH	Width of a Tag field
8'd38	SW0_LOW	Low bit of Way 0 State
8'd40	SW1_LOW	Low bit of Way 1 State
8'd2	STATE_WIDTH	How wide is the state
8'd42	SW0_OWN_POS	Does the assoc proc OWN this block?
8'd43	SW1_OWN_POS	Does the assoc proc OWN this block?
8'd44	ECC_LOW	ECC bits

7.17.19 SPCL Address Request Fields

Defines

SPCL_ADDR

Constant	Mnemonic	Definition
8'd3	ADDR2_LOW	Low bit of ADDR2 field
8'd5	ADDR2_WIDTH	ADDR2 Field Width
8'd16	ADDR1_LOW	Low bit of ADDR1 field
8'd4	ADDR1_WIDTH	ADDR1 Field Width
8'd20	BSN_LOW	Destination Bus Stop Number low bit
8'd4	BSN_WIDTH	Destination BSN Field Width

7.17.20 SPCL CSW Command Fields

Defines

SPCL_CMD

Constant	Mnemonic	Definition
8'd3	ADDR2_LOW	Low bit of ADDR2 field
8'd5	ADDR2_WIDTH	ADDR2 Field Width
8'd16	ADDR1_LOW	Low bit of ADDR1 field
8'd4	ADDR1_WIDTH	ADDR1 Field Width
8'd8	DAT0_LOW	Low byte of Data low bit
8'd8	DAT0_WIDTH	Low data width
8'd20	DAT1_LOW	Rest of DAT field
8'd16	DAT1_WIDTH	Width of upper data field

7.18 Registers

7.18.1 Cache Probe Control Register

The cache probe registers are used to generate a L2 intervention into the L1, by request of the local code. This is implemented only in the verification model, for testing purposes.

Register

R_CacxProbeCtlMagic

Attributes

-noregtest -noregdump

Address

0x00_0400 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31	Done	R	0		Intervention valid. Cleared on writing the Prb bit, set when the intervention has completed.
30	Hit	R	0		Intervention resulted in hit.
29	Dirty	R	0		Intervention resulted in dirty.
28	Lock	R	0		Intervention resulted in locked return.
					Reserved
27	IOHoldoff	RW	0		Inhibit IO write acks until this prbe has been acknowledged.
26:1	Delay	RW	0		Probe delay. Wait this number of cycles after _Prb bit is set before creating the probe.
0	Prb	RW	0		When written one, create a probe as specified.

7.18.2 Cache Probe Address Register

The cache probe registers are used to generate a L2 intervention into the L1, by request of the local code. This is implemented only in the verification model, for testing purposes.

Register

R_CacxProbeAddrMagic

Attributes

-noregtest -noregdump

Address

0x00_0404 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31:3	AddrL	RW	0		Address Low. Address[31:3] to generate probe to. Verification implementaion only.
2:0	AddrH	RW	0		Address High. Address[34:33] to generate probe to. Verification implementaion only. [35] is always zero.

7.18.3 Cache Probe Random Address Registers

The cache probe registers are used to generate a L2 intervention into the L1, by request of the local code. This is implemented only in the verification model, for testing purposes.

Register

R_CacxProbeRandAddrMagic[7:0]

Attributes

-noregtest -noregdump

Address

0x00_0500-0x00_053F (plus base address) (Add 0x8 per entry)

Bit	Mnemonic	Access	Reset	Type	Definition
36	Enable	RW	0		Send probes to the address contained in Addr
35:5	Addr	RW	0		Address. Address[35:0] to generate probe to. Verification implementation only. In ICE9A, bits [4:0] is ignored and treated as 0, since all probes are aligned to L1 cache blocks. Starting in ICE9B, bits [5:0] is ignored and treated as 0, since all probes are aligned to L2 cache blocks.

7.18.4 Cache ECC Injection Register

Controls BFM backdoor ECC injection to L1 I and D cache RAMs. This is implemented only in the verification model, for testing purposes.

Register

R_CacxInjEccMagic

Attributes

-noregtest -noregdump

Address

0x00_0408 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
1	FlipAllLinesSoon	RW	0		Flip one randomly selected bit in every cache block
0	StartRandomFlips	RW	0		Start continuous random L1 parity / ecc single-bit flipping

7.18.5 I/O Addresses in L2 Segment

Defines

CAC_IO

Constant	Mnemonic	Definition
36'hE_9000_0000	WTIOADDR	I/O writes are implemented as WTIO command, RDIO command, then data. When the RDIO is sent back to the initiator, the Addr must be set to CAC_IO_WTIOADDR.

7.18.6 Interrupt Cause Register

Register

R_CacxIntCr[7:0]

Attributes

-kernel

Address

0x00_0000-0x00_003F (plus base address) (Add 0x8 per entry)

Bit	**Mnemonic**	**Access**	**Reset**	**Type**	**Definition**
63:10		R	0		Reserved. Read as zero
9	ACTIVE	RW1C	0		If read as 1, correspoinding interrupt is asserted. Write 1 to clear. Note when clearing _Active, the _Overflow bit is also cleared, see bug3343.
8	OVERFLOW	RW1C	0		Interrupt Cause Register Overflow.
7:0	CAUSE	R	0		Interrupt Cause

When the Interrupt Cause register is over-written (that is, on the arrival of an ICR write or INT command from the CSW for an ICR whose ACTIVE bit is set) the OVERFLOW bit will be set, and all other bits will be left unchanged.

Writing 1 to ACTIVE will clear ACTIVE. Writing 1 to OVERFLOW will clear OVERFLOW. A write to either bit will leave CAUSE as it was.

7.18.7 Interrupt Delivery Register

Register

R_CacxIntDel

Attributes

-kernel

Address

0x00_1000

Bit	Mnemonic	Access	Reset	Type	Definition
63:16		R	0		Reserved.
15:12	DEST	W	0		Bus stop number of target segment.
11:8	ICRIDX	W	0		Index into target segments ICR set.
7:0	CAUSE	W	0		Interrupt Cause

7.18.8 Slow Interrupt Selection Register

Register

R_CacxSlIntSel

Attributes

-kernel

Address

0x00_00C8 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
12	CswUnCorEccIntEn	RW	0		Uncorrectable CSW ECC Interrupt is passed on to processor INT[3] IRQ5
11	CswCorEccIntEn	RW	0		Correctable CSW ECC Interrupt is passed on to processor INT[3] IRQ5
10	L2UnCorEccIntEn	RW	0		Uncorrectable L2 ECC Interrupt is passed on to processor INT[3] IRQ5
9	L2CorEccIntEn	RW	0		Correctable L2 ECC Interrupt is passed on to processor INT[3] IRQ5
8	LACSlIntEn	RW	0		Assertion of LAC (OCLA) Slow Interrupt is passed on to processor INT[3] IRQ5
7	PMISlIntEn	RW	0		Assertion of PMI Slow Interrupt is passed on to processor INT[3] IRQ5
6	SCBSlIntEn	RW	0		Assertion of SCB Slow Interrupt is passed on to processor INT[3] IRQ5
5	FLSlIntEn	RW	0		Assertion of Fabric Link Transciever Interrupt is passed on to processor INT[3] IRQ5
4	DMASlIntEn	RW	0		Assertion of DMA Slow Interrupt is passed on to processor INT[3] IRQ5
3	FSWSlIntEn	RW	0		Assertion of FSW Interrupt is passed on to processor INT[3] IRQ5
2	UARTSlIntEn	RW	0		Assertion of UART Interrupt is passed on to processor INT[3] IRQ5
1	COHESlIntEn	RW	0		Assertion of COHE Interrupt is passed on to processor INT[3] IRQ5. COHE asserts this interrupt on occurrence of an ECC error or DDR Calibration Timeout.
0	COHOSlIntEn	RW	0		Assertion of COHO Interrupt is passed on to processor INT[3] IRQ5. COHO asserts this interrupt on occurrence of an ECC error or DDR Calibration Timeout.

7.18.9 Slow Interrupt Status Register

For more details, see the “Interrupts, Again” section of the Processor Segments chapter, (section 6.19.6).

Register

R_CacxSlIntStat

Attributes

-kernel

Address

0x00_00D0 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
12	CswUnCorEcc	RW1C	0		Uncorrectable ECC detected on transfer from CSW
11	CswCorEcc	RW1C	0		Correctable ECC detected on transfer from CSW
10	L2UnCorEcc	RW1C	0		Uncorrectable ECC detected on transfer from L2 Cache
9	L2CorEcc	RW1C	0		Correctable ECC detected on transfer from L2 Cache
8	LACSlInt	R	0		LAC (OCLA) Slow Interrupt asserted
7	PMISlInt	R	0		PCI/PMI Slow Interrupt asserted
6	SCBSlInt	R	0		SCB Slow Interrupt asserted
5	FLSlInt	R	0		Fabric Link Transciever Interrupt asserted
4	DMASlInt	R	0		DMA Slow Interrupt asserted
3	FSWSlInt	R	0		FSW Interrupt is asserted
2	UARTSlInt	R	0		UART Interrupt is asserted
1	COHESlInt	R	0		COHE Interrupt is asserted
0	COHOSlInt	R	0		COHO Interrupt is asserted

7.18.10 L2 Cache ECC Mode Register

Register

R_CacxEccMode

Attributes

-kernel

Address

0x00_0100 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
5	L2TagDetEna	RW	0		Enable ECC Error Detection on L2 tag accesses
4	L2TagCorEna	RW	0		Enable ECC Error Correction on L2 tag accesses
3	CswDetEna	RW	0		Enable ECC Error Detection on CSW transfers
2	CswCorEna	RW	0		Enable ECC Error Correction on CSW transfers
1	L2DetEna	RW	0		Enable ECC Error Detection on L2 transfers
0	L2CorEna	RW	0		Enable ECC Error Correction on L2 transfers

7.18.11 L2 Cache ECC Test Register

Register

R_CacxEccTestDat

Attributes

-noregtestcpu_wr -kernel

Address

0x00_0108 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
5	L2DrvBadTag1	RW	0		Flip bit 1 of all future addresses written to the L2 Tag array
4	L2DrvBadTag0	RW	0		Flip bit 0 of all future addresses written to the L2 Tag array
3	CswDrvBadDat1	RW	0		Flip bit 1 of all words written to the CSW via IO write or cache block displacement.
2	CswDrvBadDat0	RW	0		Flip bit 0 of all words written to the CSW
1	L2DrvBadDat1	RW	0		Flip bit 1 of all even words for all future 32 byte blocks written into the L2 data array from L1 writebacks.
0	L2DrvBadDat0	RW	0		Flip bit 0 of all even words for all future 32 byte blocks written into the L2 data array from L1 writebacks.

7.18.12 L2 Cache Status Register

Register

R_CacxEccStat

Attributes

-kernel

Address

0x00_0110 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
8	L2TagMultErr	RW1C	0		Multiple ECC errors have occurred on an L2 tag lookup. Write 1 to clear.
7	L2TagCorErr	RW1C	0		Correctable error detected on an L2 tag lookup. Write 1 to clear.
6	L2TagUncorErr	RW1C	0		Uncorrectable error detected on an L2 tag lookup. Write 1 to clear.
5	CswMultErr	RW1C	0		Multiple ECC errors have occurred on a CSW transfer. Write 1 to clear.
4	CswCorErr	RW1C	0		Correctable error detected on a CSW transfer. Write 1 to clear.
3	CswUncorErr	RW1C	0		Uncorrectable error detected on a CSW transfer. Write 1 to clear.
2	L2MultErr	RW1C	0		Multiple ECC errors have occurred on an L2 transfer. Write 1 to clear.
1	L2CorErr	RW1C	0		Correctable error detected on an L2 transfer. Write 1 to clear.
0	L2UncorErr	RW1C	0		Uncorrectable error detected on an L2 transfer. Write 1 to clear.

7.18.13 L2 Cache Data ECC Error Address Register

This register gets loaded on the first ECC error signaled by either the DATA array ECC checkers.

Register

R_CacxL2EccAddr

Attributes

-kernel

Address

0x00_0118 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
35:3	ErrAddr	R	0		Address of word for first detected ECC error in L2 Cache
2:0		R	0		Reserved.

7.18.14 CSW ECC Error Address Register

This register gets loaded on the first ECC error signaled by the CSW ECC checker. It is cleared when the corresponding correctable or uncorrectable error bit is cleared.

Register

R_CacxCswEccAddr

Attributes

-kernel

Address

0x00_0120 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
35:3	ErrAddr	R	0		Address of word for first detected ECC from CSW transfer
2:0		R	0		Reserved.

7.18.15 L2 Cache Tag ECC Error Address Register

This register gets loaded on the first ECC error signaled by the Tag ECC checker. It is cleared when the corresponding correctable or uncorrectable error bit is cleared.

Register

R_CacxTagEccAddr

Attributes

-kernel

Address

0x00_0128 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
35:3	ErrAddr	R	0		Address of word for first detected ECC from a Tag lookup
2:0		R	0		Reserved.

7.18.16 L2 Cache ECC Error Syndrome Register

Each syndrome field is only meaningful if the corresponding correctable/uncorrectable error bit is set.

Register

R_CacxEccSynd

Attributes

-kernel

Address

0x00_0130 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
15:8	CswSyndHi	R	0		Syndrome from the high word of a CSW transfer.
7:0	CswSyndLo	R	0		Syndrome from the low word of a CSW transfer.

The syndrome is only captured for ECC errors from CSW transfers. (This gives us insight into which bits are failing on DIMMs. This is more valuable than knowing which bits are failing in on-chip RAMs. The register is loaded on the FIRST detected CSW ECC error after the CorErr and UnCorErr bits have been cleared.

7.18.17 L2 Cache Send SPCL Request Address Range

The SPCL addresses must span a range of 16 maximum size physical pages (64kB), so that each page can be mapped by the kernel into a separate user process. To send a SPCL, the program does a store instruction to an address in the SPCL request address range. The address of the store, and the data that is stored, are combined to produce the value that is driven onto the CSW Address bus along with the SPCL command. The CSW address encoding is described in detail in section 7.10.6.

Note that these addresses must be on separate physical pages from all other local CAC control registers as these will be accessible from user mode programs.

Register

R_Spcl[0x3F_FFFF:0]

Attributes

-noregtest -kernel

Address

0xE_BE00_0000-0xE_BEFF_FFFC

Bit	Mnemonic	Access	Reset	Type	Definition
23:0	SpclData	W	0		Data to be delivered to DMA engine via SPCL command.

7.18.18 Coherence Engine ECC Mode Register

Register

R_CohxEccMode

Attributes

-kernel

Address

0x00_0000 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
2	DetDblEna	RW	0		Enable ECC Error Detection on tag lookups. When asserted, any detected double bit error will trigger a slow interrupt from this coherence widget. (See 7.18.8.)
1	DetSnglEna	RW	0		Enable ECC Error Detection on tag lookups. When asserted, any detected single bit error will trigger a slow interrupt from this coherence widget. (See 7.18.8.)
0	CorEna	RW	0		Enable ECC Error Correction on tag lookups

Programmer's note: **Bugzilla 1990 finds that the behavior of the COH when CorEna is clear could be unpredictable when an ECC error is detected in a master tag array. For this reason, the CorEna bit should always be set to 1 when the COH is in use.**

7.18.19 Coherence Engine ECC Test Register

Register

R_CohxEccTestDat

Attributes

-kernel

Address

0x00_0018 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
1	DrvBadDat1	RW	0		Flip bit 1 of word 0 in any tag written into any tag array
0	DrvBadDat0	RW	0		Flip bit 0 of word 0 in any tag written into any tag array

7.18.20 Coherence Engine ECC Status Register

Register

R_CohxEccStat

Attributes

-kernel

Address

0x00_0020 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
2	MultErr	RW1C	0		While either CorErr or UnCorErr was set, a subsequent ECC (single or double) error was detected. Write 1 to clear.
1	CorErr	RW1C	0		Correctable error detected on a tag lookup. Write 1 to clear. If this bit and the DetEna bit in the CohxEccMode register are both set, the Coh will send a slow interrupt to each processor segment. One or more tag arrays may have reported a single bit error in a given cycle.
0	UncorErr	RW1C	0		Uncorrectable error detected on a tag lookup. Write 1 to clear.

Note that MultErr is NOT asserted if two or more TAG arrays report an ECC error in the *same* cycle. MultErr is only asserted if a new ECC error occurs while CorErr or UncorErr is alread asserted.

7.18.21 Coherence Engine ECC Error Address Register

This register gets loaded on the first ECC error signaled by the CSW ECC checker. It is updated only if CorErr and UncorErr are both clear.

Register

R_CohxEccAddr

Attributes

-kernel

Address

0x00_0028 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
30:3	ErrAddr	R	0		Address <34:7> of Block for first detected ECC tag lookup
2:0	Array	R	0		Which tag array had the problem? If multiple arrays reported an error, the lowest numbered array is reported here.

7.18.22 Twice9+ Coherence Engine ECC Error Address Register

For Twice9+ this new 64 bit SCB register gets loaded on the first ECC error signaled by the CSW ECC checker. It is updated only if CorErr and UncorErr are both clear.

Register

R_CohxEccAddrTwcPlus

Attributes

-kernel -Product=TWC9A+

Address

0x00_0050 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Product	Definition
47:7	ErrAddr	R	0		TWC9A+	Address <47:7> of block for first detected tag lookup ECC error. Some number of MSB bits are padded with zeros depending on the design revision.
6:0	Array	R	0		TWC9A+	Identifies which tag array had the problem. If multiple arrays reported an error, the lowest numbered array is reported here. MSBs are padded with zeros depending on the number of tag arrays in the specific design revision.

7.18.23 Coherence Engine ECC Error Syndrome Register

Register

R_CohxEccSynd

Attributes

-kernel

Address

0x00_0040 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
7:0	ErrSyndrom	R	0		Syndrome of first detected ECC error from Master Tag Lookup

7.18.24 Coherence Engine Active Processor Segment Register

Register

R_CohxNumSegs

Address

0x00_0048 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Product	Definition
6:3	ActiveSegCountTwc	RW	10		TWC9A+	Number of L2 Segments currently enabled for oper Must be either 1 or 10.
2:0	ActiveSegCount	RW	6			Number of L2 Segments currently enabled for oper Must be either 1 or 6.

The NumSegs register allows the chip to be configured as a uniprocessor, if necessary. The value in this register must be set prior to initial program load. The value from this register is loaded into the appropriate INVDONE counter whenever the COH sends out a PRBINV request on behalf of a processor or PMI device. (A transaction that causes a PRBINV is not complete until all active L2 segments have sent an INVDONE signal to the appropriate COH. See section 7.12.3.5.)

7.19 Register Allocation

This chapter instantiates the six copies, plus the local copy of CAC registers. It also instantiates the two sets of COH control registers.

7.19.1 CacLoc

Register

R_CacLoc* : R_Cacx*

Address

0xE_9E00_0000-0xE_9EFF_FFFF

7.19.2 Coho

Register

R_Coho* : R_Cohx*

Address

0xE_0000_0000-0xE_00FF_FFFF

7.19.3 Cohe

Register

R_Cohe* : R_Cohx*

Address

0xE_0900_0000-0xE_09FF_FFFF

Chapter 8

Memory Controller

[Last modified $Id: memctl.lyx 50693 2008-02-07 16:01:46Z wsnyder $]

8.1 Overview

The ICE9 chip has two built-in memory controllers, each of which interfaces to one 1-GB, 2-GB, 4-GB, or 8-GB 72-bit DDR2 SDRAM DIMM. The chip accomodates memory clock rates of 267, 333, and 400 MHz, corresponding to data rates of 533, 667, and 800 MHz, respectively.

The memory controller functionality is partitioned accross two functional units, DDR and DDP. The DDP unit contains the DDR2-PHY, which is implemented as a hard IP macro (purchased from Esilicon). The DDR Unit is composed of the following subsections:

1. DDI - Interface block between the DDR2 Controller (DDC) and the Coherence Controller (COH). Designed by SiCortex.
2. DDC - DDR2 SDRAM controller IP logic block. Purchased source code from Northwest Logic and synthesized.
3. DDD - Read datapath interface to DDR2 PHY

The two instances of the DDR unit are referred to as the “even” and ’odd” DDR units. The “even” DDR instance is ddre (sometimes called ddr0), while the “odd” instance is ddro (sometimes referred to as ddr1). The even instance is on the east side of the die. The instances are distinguished by the static input pin tie_ddrx_id (for ddre/ddr0 it is tied to 1’b0(GND), while for ddro/ddr1 it is tied to 1’b1 (VDD)). The two instance of the DDP unit are similarly named, however their is no need for a static signal to distinguish them since a given instance of DDP does not need to know whether it is the “even” or “odd” instantiation.

8.2 Differences, Bugs, and Enhancements

8.2.1 Product and Chip Pass Differences

1. ICE9B fixes the DDR unit to support IO driver calibration before the DRAM initialization sequence, bug2276. In ICE9A the Ddr/Ddp units currently only support updating values into the IMP_P_HV[3:0] and IMP_N_HV[3:0] inputs of the DDR2 IO cells during one of the mission mode time CalModes. When SoftReset is asserted the PHY puts in default strong values (low impedence biased) into these.
2. ICE9B fixes some of the ODT on/off range values, bug2401. The NWL controller was supposed to support the following range of ODT turn on/off times for Ice9a’s DDR-Phy: ON time range: controlled by DdrxPhyCfg2_AsicDqsOdtOn and DdrxPhyCfg2_AsicDqOdtOn -2.5 clocks <-> 0 clocks (in half cycle increments) relative to the start of the read preamble OFF time range: controlled by DdrxPhyCfg2_AsicDqsOdtOff and DdrxPhyCfg2_AsicDqOdtOff -1.5 clocks <-> 2 clocks (in half cycle increments) relative to the start of the read preamble. However, the bug causes the -2.5 and -2 clocks turn on times to NOT work with turn off times of 1.5 and 2 clocks.
3. TWC9A fixes access to any SCB bus slave hanging while the DDR controller is in reset, bug2928.

4. NEED IMPL: TWC9A drops support for unbuffered DIMMs.

8.2.2 Known Bugs and Possible Enhancements

1. Calibration Mode 2 can cause Ddi to hang waiting for Powerdown, see bug2013. When setting AutoCalUpdate in cal mode 2 (update during prechargePowerdown), the Ddi can hang. This is caused when a request is at the head of the queue requesting to be sent to the controller at the time we start the calibration update process. The calibration logic spins in place waiting for powerdown entry. However, this pending request causes the powerdown counter to be cleared on every cycle, which blocks the Ddr from ever entering powerdown mode. To workaround, do not use calibration mode 2.

2. The DDR bank address could be changed to better optimize page hits, bug2068.

8.3 General Description

8.3.1 Clocks

The memory interface has two clock domains: CCLK and DCLK. CCLK is the same clock used on the core side of DDR (COH and CSW units) and logic which runs on the DCLK which is same clock used by DDC, the DDR2-PHY and the DDR2 SDRAM DIMMs (Note that some of the logic really runs off of DM90CLK which is a minus 90 degree shifted version of DCLK).

The required relationship between the clock is:

CCLK <= DCLK < (2 * CCLK)/1.05

Note that since DCLK (or DM90CLK) is also used for driving clocks to the DIMM and the PHY's DLLs it has the addition restriction that 125MHz <= DCLK <= 465MHz (125MHz correlates to the maximum tCK cycle time supported by target DIMMs and 465MHz is the maximum frequency supported by the True Circuits analog DLLs used in the PHY).

Table 8.1: Recommended DCLK to CCLK relationships

DCLK	CCLK
267 MHz	140 MHz - 267 MHz
333 MHz	175 MHz - 333 MHz
400 MHz	210 MHz - 400 MHz

Note that the Analog Bits PLL used on the ASIC drive out a two clocks at DCLK frequency: PLLOUT_1 and PLLOUT_2 which is shifted positive 90 degress relative to PLLOUT_1. Thus DCLK must be tied to PLLOUT_2 and DM90CLK tied to PLLOUT_1 in order to achieve the desired minus 90 degree shift.

8.3.2 Reset and Initialization

Startup sequence for the DDR interface to come up correctly.which will cause R_DdrxDdcDdpSoftReset to assert)

1. At startup, power will be brought up for the ICE9 and for the DIMMs (in accordance with JEDEC standard JESD79-2B 2.3.1a (page 9)).

2.Global reset will be asserted from before the start of power-up and kept asserted during power-up. (This is to address the JEDEC mandate of attempting to maintain CKE below 0.2*VDDQ and ODT at a low state during power-up (they are asynchronously pulled low when reset is asserted).

3. The dclk resets (reset_e1der_l and reset_e1dor_l) must remain asserted for at least 1us after the power ramp has been completed. (This is a requirement of the analog DLLs used in the DDR2-PHY).

4. After all clocks are appropriately configured and stable (at least those relevent to memory operation: cclk, d0clk, d1clk, d0m90clk, d1m90clk) deasset the cclk and dclk resets.

HERE NEED TO ADD CONFIGURING OF CK IO DRIVE STRENGTH THEN RELEASE THE RESET FOR THE CLOCK FLOPS

5. The deassertion of the dclk resets will cause clocks to be driven to the DIMMs (JEDEC requires a min of 200us of stable clock, some or all of which can be satisfied in the shadow of steps 6 -> 11, which would reduce the delay value required by R_DdrxDdcMemCfg3_Delay).

Note 5-1: The inital value of R_DdrxDdcDdpSoftReset will keep the memory controller and DDR2-PHY IP blocks (DDC/DDP) in reset.

Note 5-2: The initial value of R_DdrxDdiMemLoopBack we be such that any memory references received by the DDR units will be looped back such that they receive completion notification.

6. Write a 0 to R_DdrxDdpDLLReset to deassert reset to the PHY DLLs.

Note 6-1: The minimum total assertion time of R_DdrxDdpDLLReset is 1us after the power ramp completes (clocks need to be stable for at least a few cycles before this reset is deasserted).

Note 6-2: After R_DdrxDdpDLLReset is deasserted no reads can go out to memory for 500 cycles while the DLLs are possibly unlocked.

7. Based on data obtained from the DIMMs Serial Presence Detect through the on die I2C Master Controller and from data on the DIMM configuration provided via the Module Service Processor (MSP), the boot processor will then write the CSR registers R_DdrxDdcMemCfg1-5, R_DdrxDdcDIMMODT, R_DdrxDdpODT, and R_DdrxDIMMSize via the SCB bus. The boot processor may also write the registers R_DdrxDdiMifCfg1-2 and R_DdrxDddRdDelay, otherwise the defaults will be used (R_DdrxDdiMifCfg1-2 can be modified via the SCB at runtime also).

8. Write appropriate values to R_DdrxPhyCfg1-3 and R_DdrxDddRdDelay if the defaults prove inadequate.

9. The values of R_DdrxDdpDLLLane0-8 will need to be set. This step can be satisfied with known good values or some values which be adjusted as decribed in the section below "PHY Read Path DLL Calibration".

10. The boot processor will them write a 0 to R_DdrxDdcDdpSoftReset to deassert the soft reset to DDC and DDP.

11. After the boot processor has insured that there are no outstanding read or write requests (i.e. no TIDs are in flight (this may involve some sequence of memory ordering directives)), it will then write a 0 to R_DdrxDdiMemLoopBack.

12. Once the DDC / DDP soft reset is deasserted, DDC will begin issuing an initialization sequence compliant with the JEDEC standard, and DDI will begin queuing up read and write requests.

13. Issue a memory test sequence (note that failure of the memory test must not be considered a fatal startup error such that it blocks testing to calibrate the PHY DLLs.

14. Clear memory (write 0s to all locations).

8.3.3 Serial Presence Detect

DDR2 SDRAM memory DIMMs interfacing to ICE9 must implement Serial Presence Detect in accordance with JEDEC Standard No. 21-C. Details discussed herein (in particular the SPD byte #s address mapping), reference the preliminary publication of "Appendix X: Serial Presence Detects for DDR2-SDRAM (Revision 1.2).

On the board, the even side DIMM (on the east side of the chip and interfacing to ddre / ddr0) will be hard coded with its SDA[2:0] inputs tied to 000, resulting in an I2C address of 0x50, while the odd side DIMM will have its SDA[2:0] inputs tied to 001, resulting in an I2C address of 0x51.

8.3.4 PHY Read Path DLL Calibration

For detailed structural information on the DLL used in the PHY, see the corresponding subsection of the "DDP Unit - DDR2 SDRAM PHY IP Block" section of this specification. This section describes the process for software to figure out optimal DLL settings for each of the 9 byte lanes of each DDR interface. The process is for software to sweep through DLL settings, doing a read with each value, to figure out an eye window. The center of the eyes will point to the best DLL settings. There are a number of issues which need to be addressed with software and hardware support:

1. The processor running the software doesn't see the ECC. To address this, the hardware includes CSRs which capture the ECC value of data transfered in association with read requests to memory.

2. An incorrect DLL settings can result in the PHY not returning any read data. To deal with this the hardware has a mode (controlled with R_DdrxDdiRdTimeOutAutoComplete) to prevent hanging. Some clean up of internal state is required before the next read access attempt (controlled with R_DdrxDdiRdPathRst).

3. An incorrect DLL setting can result in the PHY returning incomplete read data. R_DdrxDdiRdPathRst is used in between read attempts to insure the read datapath is returned to a known good state before attempting the next read.

8.3.4.1 Overview of DLL calibration process

Since each byte lane has two DLLs, the basic idea is to fix the DLL setting for one of the DLLs (referred to as the reference DLL). Do a number of reads as the other DLL is sweeped across a range which is expected to include its eye. Then change the reference DLL and again do reads while sweeping the other DLL. Since each DLL has 160 steps, it would take alot of reads to sweep the entire space. We can reduce the search window because we know that Slave1 will need to be close to 1/4 of the reference cycle. Based on the analysis provided in the DLL subsection of the DDP Unit section of this specification, it is recommended that Slave1 be used as the reference DLL, and it should sweep from 0 to 38. The Slave 0 DLL needs to sweep a range which covers the min to max trace length delay for byte lanes. The Slave 0 sweep range is recommended to be 1-134.

8.3.4.2 DLL Calibration flow

Suggested DLL calibration flow (Note that these steps need to be executed for both of DDR interfaces. Total calibration time can be reduced by doing them in parallel, but care should be taken to insure they don't alias to the same address in any of the cache levels).

1. Go though reset and initalization sequence as discussed above.

2. Set R_DdrxDdiECCCaptureEnable_EnableRdECCCapture

3. Set R_DdrxDdiRdTimeOutAutoComplete_Enable CSR to enable auto completion of reads that hang.

4. Issue a write of a signature pattern such that the write is pushed all the way to DRAM.

Note 4-1 The signature should be chosen carefully so that the each of the 9 byte lanes recieves unique data over the 8 bursts of the read returned from the DIMM. Especially note we want the 8 burst of the ECC to be unique also, so that pattern accross each 8B chunk should factor that in.

5. Wait for the write to complete (TID is released).

6. Issue a read to the same address as the previous write, such that the read is issued all the way to DRAM.

7. A few cycles after the read data is driven onto the CSW bus, copies of the ECC bits are written into the CSRs R_DdrxDdiRdECCCapture0-1.

8. Wait for read data to return to the processor.

9. Compare the read data with the expected value. Use the SCB bus to access R_DdrxDdiRdECCCapture0-1. A byte lane must compare correctly for all of its 8 transfer bursts.

10. Check R_DdrxDdiRdTimeOutAutoComplete_RdHang, if it is set then interpret this to mean that all the byte lanes failed for the given set of DLL settings.

11. Based on the results of steps 9 and 10 log the sucess/failure result for each of the 9 byte lanes.

12. Write new values to R_DdrxDdpDLLLane0-8_Slave0Adj and possibly R_DdrxDdpDLLLane0-8_Slave1Adj. (according to the DLL spec it takes "a couple of cycles" for the DLL to operate glitch free at the new settings, the time to execute steps 13-15 should more than account for this).

13. Write a 1 to R_DdrxDdiRdPathRst.

14. Write a 0 to R_DdrxDdiRdPathRst.

15. Clear R_DdrxDdiRdTimeOutAutoComplete_RdHang (it is W1C).

16. Loop back to step 6.

8.3.5 DIMM Requirements

1. 240 Pin DDR2 SDRAM Unbuffered or Registered DIMM.

2. x72 DIMM (72 total data pins, 64 data plus 8 check bits (referred to as ECC DIMMs).

3. DRAM chips on the DIMM are x8 chips (9 on single rank DIMMs, 18 on dual rank DIMMs), and are one of the following sizes: 512 Mb, 1 Gb, 2 Gb, or 4 Gb. Note that this implies the chips have 4 or 8 banks (2 or 3 bank address bits) and 10 column address bits.

4. Transfer rate requirement: 266, 333, or 400 MHz tCK. Note 266 MHz may not be supported in systems where CCLK is faster than 266 MHz.

Table 8.2: Supported memory configurations per DDR interface (half of the total main memory connected to each ICE9 chip).

Note that 4 rank configurations are not targeted because the DDR2-PHY is not designed to operate at full speed with the loading of a 4 rank configuration.

DIMM Configuration	DRAM chips	Target Configuration
1GB (2 rank) *	18-512Mb (64Mx8) chips	YES
1GB (1 rank)	9-1Gb (128Mx8) chips	YES
2GB (1 rank)	9-2Gb (256Mx8) chips	YES
2GB (2 rank)	18-1Gb (128Mx8) chips	YES
4GB (1 rank)	9-4Gb (512Mx8) chips	NO
4GB (2 rank)	18-2Gb (256Mx8) chips	YES
4GB (4 rank)	36-1Gb (128Mx8) chips	YES
8GB (2 rank)	18-4Gb (512Mx8) chips	NO
8GB (4 rank)	36-2Gb (256Mx8) chips	NO
16GB (4 rank)	36-4Gb (512Mx8) chips	NO

* Note that this configuration requires setting R_DdrxDdcMemCfg3_Bankbits = 0

8.3.6 Addressing

The ICE9 chip has a 64GB address space, 32GBs of which is for main memory (cacheable). Each instance of the DDR unit can interface with up to 16GB of memory (the 16GB is logically possible based on the functionality of the design, however the target maximum is 8GB because of physical design issues and the expectation that the largest DIMMs available in 2 or less rank configurations will be 8GB DIMMs in the foreseeable future). Because of the 64GB address space the address bus has 36 bits (35:0), however the DDR units drops bits for the following reasons:

1. Bit 35 is dropped because it is always 0 for main memory references.
2. Bit 6 is dropped because it is used to decide which DDR interface a request goes to

(i.e. it is always fixed for a given interface).

3. Bits 2:0 are not used because byte addressable requests are not supported by DDR2.

So for example the incoming address coh_ddr_RdAddr_c2a[35:0] becomes addr[34:7,5:3] => addr[33:3]. Addr[33:3] is the format used within the DDR unit.

The DDR section handles 64B memory references (including ECC they are 72B requests). Reads presented to the DDR unit are required to be full 64B reads. It returns the requested quadword (QW) (128 bits + 16 bits ECC) first for read requests according to Table 6.1. The read address presented to memory is QW aligned (i.e. address[3] is always driven LOW on the address send to DDC). The only supported write transaction sizes are 64B and 32B. Write requests for 64B blocks must be aligned such that the starting address is 000 (the starting address is specified by bits [5:3] of the incomming address). 32B writes are converted into 64B memory writes with the byte mask bits driven "low" to prevent updating memory for the invalid part of the transfer. Write requests for 32B blocks must be aligned such that the starting address[5:3] is 000 or 100.

Table 8.3: Data Transfer Order

Address[7:0]	Address[5:4]	Order of doublewords (DWs) out of DIMM	Data on CSW {Data0, Data1, Data2, Data3, Data4, Data5, Data6, Data7}
0x00 or 0x08	00	0,1,2,3,4,5,6,7	{0, 1, 2, 3, 4, 5, 6, 7}
0x10 or 0x18	01	2,3,0,1,6,7,4,5	{2, 3, 0, 1, 6, 7, 4, 5}
0x20 or 0x28	10	4,5,6,7,0,1,2,3	{4, 5, 6, 7, 0, 1, 2, 3}
0x30 or 0x38	11	6,7,4,5,2,3,0,1	{6, 7, 4, 5, 2, 3, 0, 1}

Table 8.5: Types of Memory writes:

Given that the order of write data arriving at DDR from CSW is:
{Data0, Data1, Data2, Data3, Data4, Data5, Data6, Data7}.
"None" => write mask bits are deasserted so that data in not overwritten in main memory.

coh_ddr_WrHalfMask_c4a	coh_ddr_WrAddr_c4a[5]	Order of data sent out to memory
'E_CohHalfMask_W64	0	{Data0, Data1, Data2, Data3, Data4, Data5, Data6, Data7}
'E_CohHalfMask_W64	1	{Data4, Data5, Data6, Data7, Data0, Data1, Data2, Data3}
'E_CohHalfMask_L32	0	{Data0, Data1, Data2, Data3, None, None, None, None}
'E_CohHalfMask_L32	1	{None, None, None, None, Data0, Data1, Data2, Data3}
'E_CohHalfMask_H32	0	{Data4, Data5, Data6, Data7, None, None, None, None}
'E_CohHalfMask_H32	1	{None, None, None, None, Data4, Data5, Data6, Data7}

8.3.7 Interface Between DDR and the Coherence Controller (COH)

Table 8.7: COH/DDR Interface

Signal name	Description
coh_ddr_RdValid_c2a	Asserted to signify a read request is being sent from COH to DDR
coh_ddr_RdAddr_c2a[35:3]	Address of a read request. Qualified by coh_ddr_RdValid_c2a
coh_ddr_RdTID_c2a[4:0]	TID of a read request. Qualified by coh_ddr_RdValid_c2a
coh_ddr_RaWShootDown_c3a	Asserted to shoot down the read which was issued one cycle earlier
coh_ddr_RdShootDown_c4a	Asserted to shoot down the read which was issued two cycles earlier.
coh_ddr_WrValid_c4a	Asserted to signify a write request is being sent from COH to DDR
coh_ddr_WrHalfMask_c4a[1:0]	See "Table 2: Types of Memory Writes" for a description of how the half mask is used. Qualified by coh_ddr_WrValid_c4a.
coh_ddr_WrAddr_c4a[35:3]	Address of write request. Qualififed by coh_ddr_WrValid_c4a
coh_ddr_WrTID_c4a[4:0]	TID of write request. Qualified by coh_ddr_WrValid_c4a
coh_ddr_Data0_c4a[71:0]	Write DW0
coh_ddr_Data1_c4a[71:0]	Write DW1
coh_ddr_Data2_c5a[71:0]	Write DW2
coh_ddr_Data3_c5a[71:0]	Write DW3
coh_ddr_Data4_c6a[71:0]	Write DW4
coh_ddr_Data5_c6a[71:0]	Write DW5
coh_ddr_Data6_c7a[71:0]	Write DW6
coh_ddr_Data7_c7a[71:0]	Write DW7
ddr_coh_WrTIDVal_c5a	Asserted when a write has been completed (safe to resue the TID)
ddr_coh_WrTID_c5a	TID of a completed write request, Qualified by ddr_coh_WrTIDVal_c5a
ddr_coh_BackPressure_c5a	Asserted if DDR can't accept anymore requests
ddr_coh_DataValid_c2a	Asserted when a read is returning data.
ddr_coh_DataTarget_c2a[8:0]	CSW target vector corresponding to read data return. Qualified by ddr_coh_DataValid_c2a
ddr_coh_RdShotDown_c2a	Asserted when a read shoot down has been completed.
ddr_coh_DataTID_c2a[4:0]	Contains the TID for either: 1. Read data returning, Qualified by ddr_coh_DataValid_c2a 2. Read which was shotdown, Qualified by ddr_coh_RdShotDown_c2a
ddr_coh_Data0_c2a[71:0]	Read DW0
ddr_coh_Data1_c2a[71:0]	Read DW1
ddr_coh_Data2_c3a[71:0]	Read DW2
ddr_coh_Data3_c3a[71:0]	Read DW3
ddr_coh_Data4_c4a[71:0]	Read DW4
ddr_coh_Data5_c4a[71:0]	Read DW5
ddr_coh_Data6_c5a[71:0]	Read DW6
ddr_coh_Data7_c5a[71:0]	Read DW7

8.4 DDI Section

8.4.1 Overview

The DDI block is the interface between the Coherence Controller (COH) and DDR2 Controller (DDC). The DDI accepts requests (read and write commands) from the COH and issues them to the DDC. DDI has two clock domains, the CCLK which interfaces with the COH, and the DCLK domain which interfaces with the DDC. All clock domain crossings are done using standard dual rank pulse synchronizers.

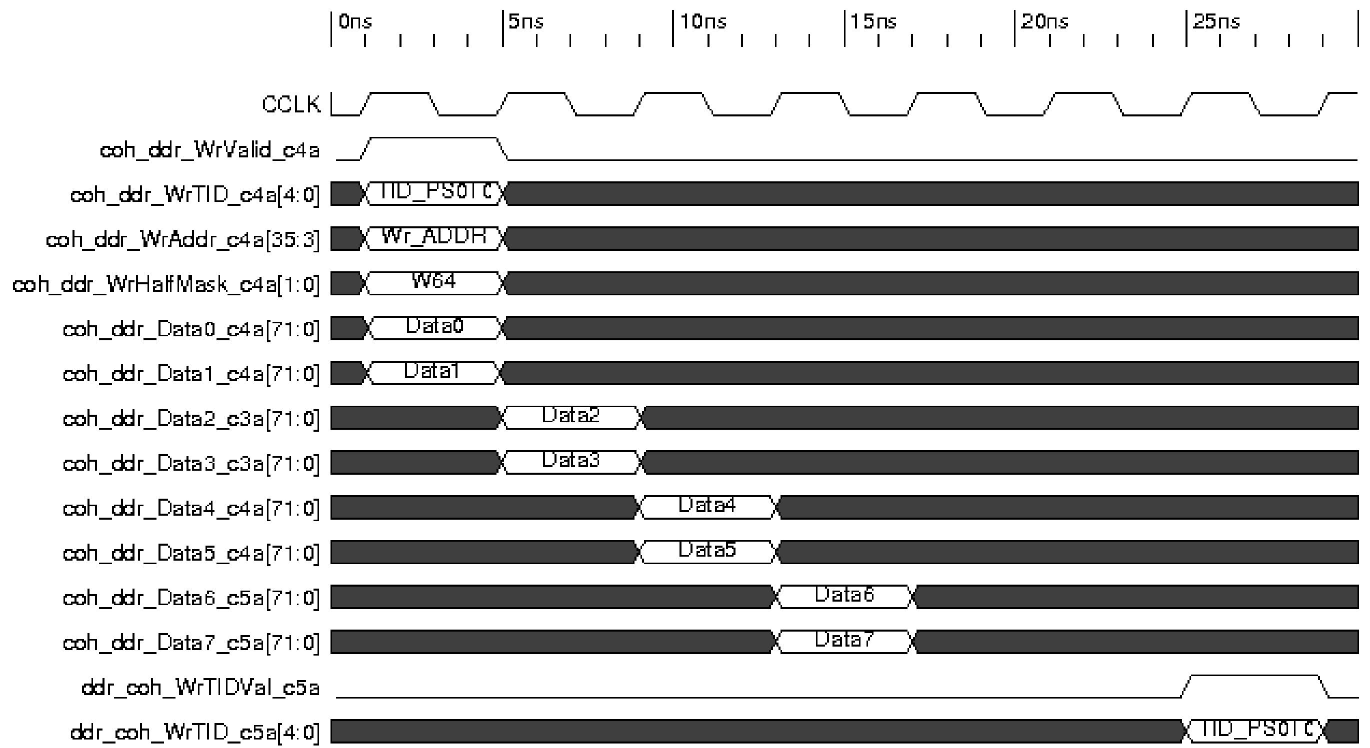


Figure 1: Write request and completion (Note there is Not a fixed time between request and completion)

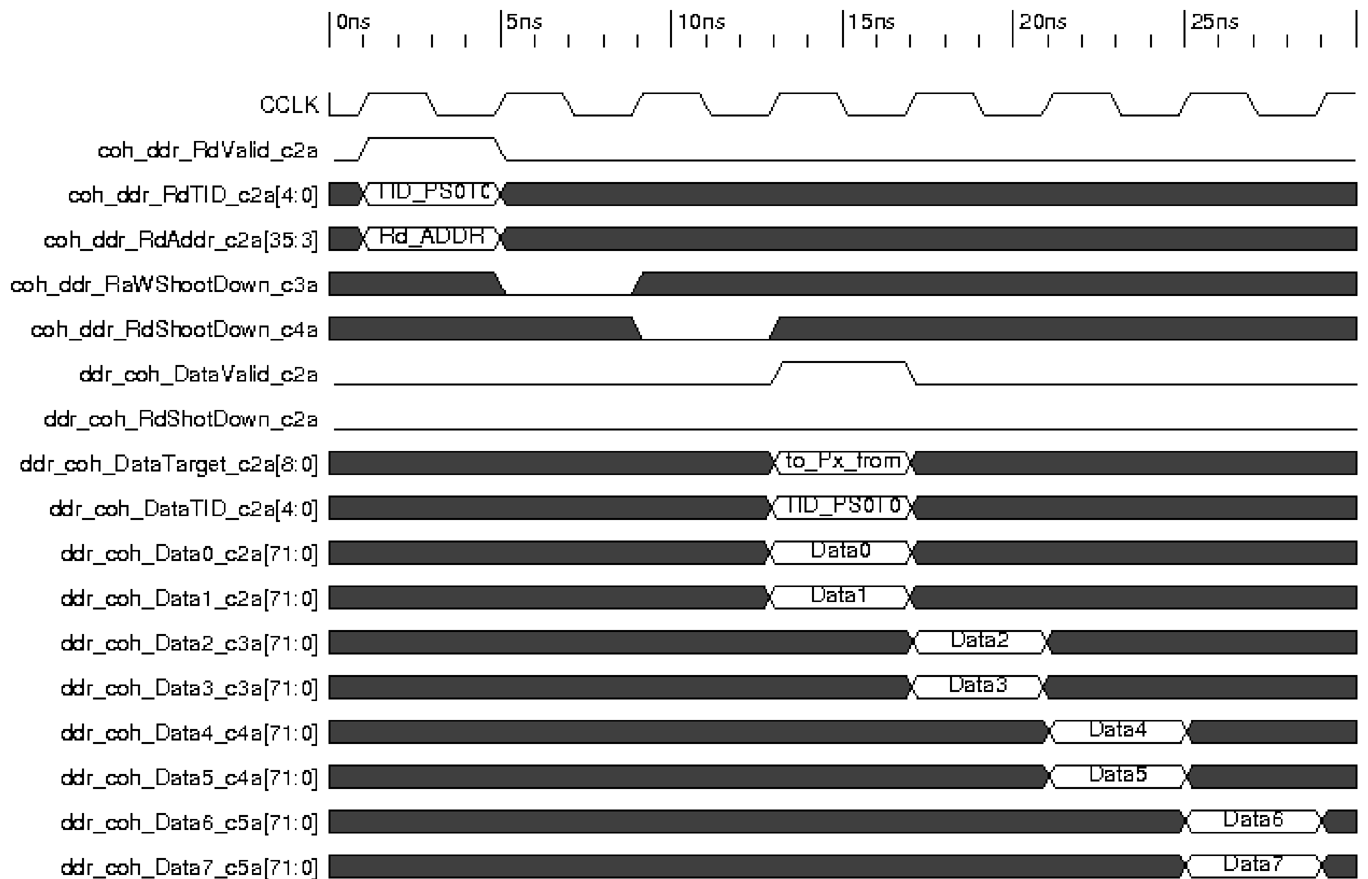


Figure 2: Read request with normal data return.
(Note this figure does not illustrate accurate delay between the arrival of the Rd request and the data return).

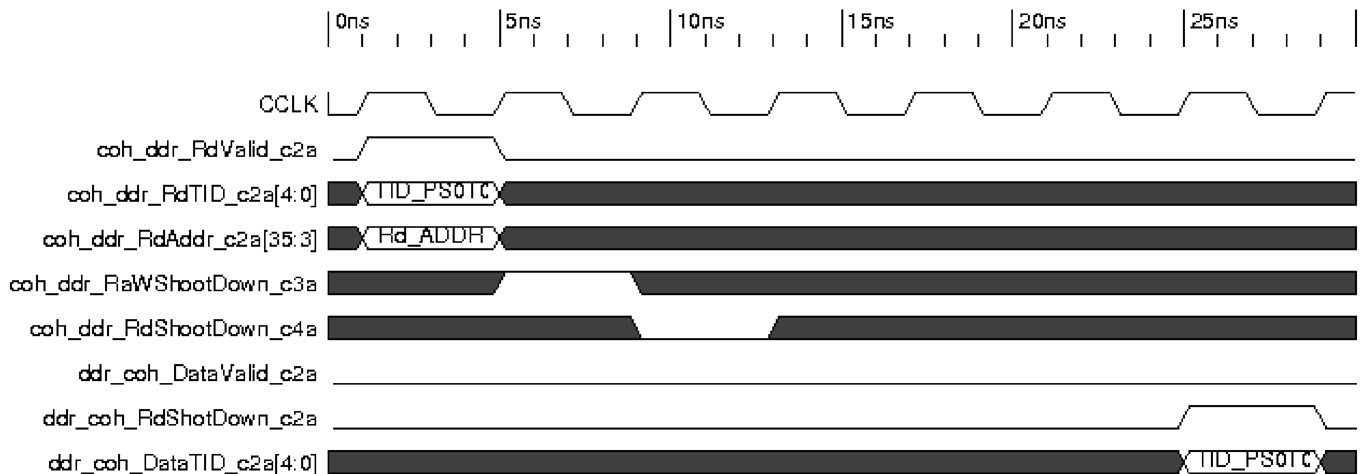
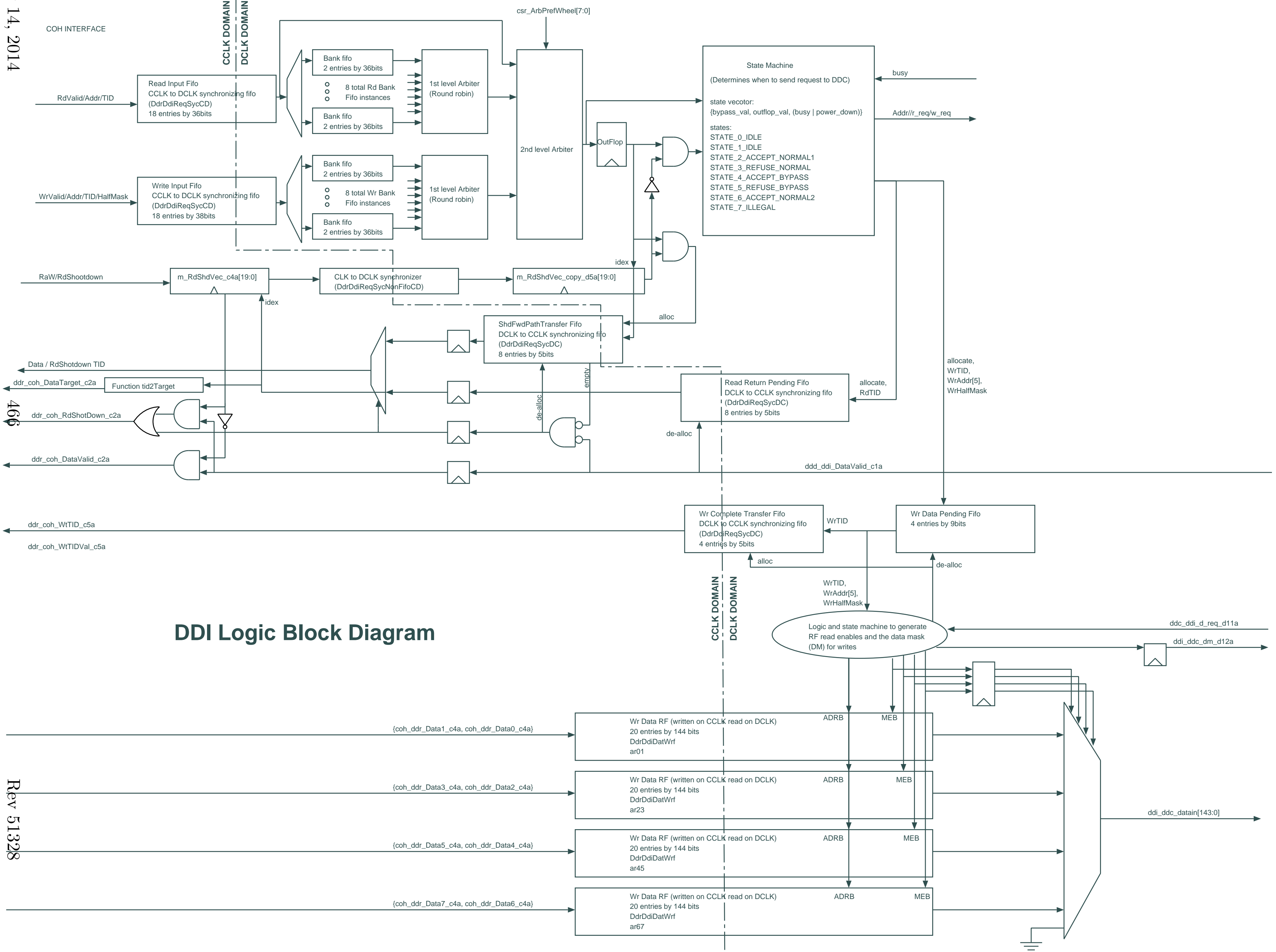


Figure 2: Read request which is shot down.
(Note 1. There is not a fixed time between the Rd request and the RdShotDown completion notification.
Note 2. The behavior is similar if coh_ddr_RdShootDown_c4a asserted or if both assert in their respective valid cycles).


DDI Logic Block Diagram
COH INTERFACE
CCLK DOMAIN
DCLK DOMAIN
csr_ArbPrefWheel[7:0]
RdValid/Addr/TID
Read Input Fifo
CCLK to DCLK synchronizing fifo
(DdrDdiReqSycCD)
18 entries by 36bits
Bank fifo
2 entries by 36bits
8 total Rd Bank
Fifo instances
1st level Arbiter
(Round robin)
WrValid/Addr/TID/HalfMask
Write Input Fifo
CCLK to DCLK synchronizing fifo
(DdrDdiReqSycCD)
18 entries by 38bits
8 total Wr Bank
Fifo instances
2nd level Arbiter
OutFlop
State Machine
(Determines when to send request to DDC)
state vecotor:
{bypass_val, outflop_val, (busy | power_down)}
states:
STATE_0_IDLE
STATE_1_IDLE
STATE_2_ACCEPT_NORMAL1
STATE_3_REFUSE_NORMAL
STATE_4_ACCEPT_BYPASS
STATE_5_REFUSE_BYPASS
STATE_6_ACCEPT_NORMAL2
STATE_7_ILLEGAL
busy
Addr//r_req/w_req
RaW/RdShootdown
m_RdShdVec_c4a[19:0]
CLK to DCLK synchronizer
(DdrDdiReqSycNonFifoCD)
m_RdShdVec_copy_d5a[19:0]
idex
alloc
ShdFwdPathTransfer Fifo
DCLK to CCLK synchronizing fifo
(DdrDdiReqSycDC)
8 entries by 5bits
Data / RdShotdown TID
ddr_coh_DataTarget_c2a
Function tid2Target
ddr_coh_RdShotDown_c2a
ddr_coh_DataValid_c2a
de-alloc
empty
Read Return Pending Fifo
DCLK to CCLK synchronizing fifo
(DdrDdiReqSycDC)
8 entries by 5bits
allocate,
RdTID
allocate,
WrTID,
WrAddr[5],
WrHalfMask
ddd_ddi_DataValid_c1a
ddr_coh_WtTID_c5a
ddr_coh_WtTIDVal_c5a
Wr Complete Transfer Fifo
DCLK to CCLK synchronizing fifo
(DdrDdiReqSycDC)
4 entries by 5bits
WrTID
Wr Data Pending Fifo
4 entries by 9bits
Logic and state machine to generate
RF read enables and the data mask
(DM) for writes
ddc_ddi_d_req_d11a
ddi_ddc_dm_d12a
{coh_ddr_Data1_c4a, coh_ddr_Data0_c4a}
{coh_ddr_Data3_c4a, coh_ddr_Data2_c4a}
{coh_ddr_Data5_c4a, coh_ddr_Data4_c4a}
{coh_ddr_Data7_c4a, coh_ddr_Data6_c4a}
Wr Data RF (written on CCLK read on DCLK)
20 entries by 144 bits
DdrDdiDatWrf
ar01
ar23
ar45
ar67
ADRB
MEB
ddi_ddc_datain[143:0]

8.4.2 Request Path

DDI can accept one read and one write command every cycle, and is structured to handle up to a total of 20 write requests and 20 read requests. Each request comes with an associated address, TID, and a valid signal. Write requests arrive coincident with the first cycle of the write data transfer. The request path for reads and writes are separate for most of DDI, allowing read and writes to pass each other (the COH prevents hazards). Incoming requests are flopped into a flop-based synchronizing fifo (one for reads and another for writes). Requests are read out of the input fifo on the DCLK and transfered to a bank fifo (based on the bank bits of the address). Since DDC is designed to manage 8 banks of memory, DDI has 8 read bank fifos and 8 write bank fifos. The head entry of the bankfifos arbitrate for access to DDC. Each cycle, a two-level arbiter selects a request to send to the DDC (if there is a valid one). The first level has parallel arbiters (one for reads, and one for writes), each of which round-robins between the valid head entries of the 8 bank FIFOs. The second level chooses which wins. The grant algorithm gives preference to reads for a fixed number of consecutive grants, then to writes for a fixed number of consecutive grants (the ratio of reads to write grant preference is set through a configuration register). In any cycle where no reads or writes are bidding from any of the bank FIFOs, the arbiter will select the head entry of the read input FIFO if it is valid. The request which wins arbitration is flopped and goes through logic to be issued to the DDC.

Refer to the DDC section for documentation and waveforms decribing the interface between DDI and DDC for issuing requests to DDC.

When the DDC accepts a write request, the write request is pushed onto the Write Data Pending Fifo where it will remain until the DDC asks for the associated write data (There is no fixed timing between when the DDC accepts a write request when it will be ready to accept the write data). When DDC asks for the write data (which is supplied by the data path logic) the entry is deallocated from the Write Data Pending Fifo so the TID of this completed write can be sent to the COH. This is the point where the COH can safely release the write from the Write Back Cam. The TID needs to be synchronized back over to the CCLK domain before sending it to the COH. This is done through the Write Complete Transfer Fifo.

When the DDC accpets a read request, the read request is pushed onto the Read Return Pending Fifo which synchronize from the DCLK to CCLK domain. The head entry is deallocated (providing the TID) when the DDD section signifies the return of read data. The Read TID is used to construct the CSW target vector.

8.4.3 Read Shoot Down

The request path incorporates logic to allow reads to be shot down. This allows the COH to issue reads speculatively to improve performance and also to kill reads which would cause a RAW hazard due to a write in DDI which has not as yet completed. By the time the shoot down signal is received in DDI, the read may be in the forward path (not yet issued to DDC) or the return path (in the Read Return Pending Fifo). Shoot down commands are logged into a vector (m_RdShdVec_c4a[19:0]), where each entry corresponds to one of the 20 possible TIDs available for read usage. When a read request is issued from COH it clears the corresponding entry in the shoot down vector, and when a shootdown is received from the COH it sets the corresponding entry. Because the TID is used to execute the shootdown, DDI cannot accept another request with the same TID until the shootdown completion has been confirmed via the ddr_coh_RdShotDown_c2a / ddr_coh_DataTID_c2a signal set.

In the forward path, reads that win arbitration for access to DDC are checked against a DCLK domain copy of the shoot down vector (m_RdShdVec_copy_d5a[19:0]) and not issued to DDC if the corresponding entry is set. Instead of entering the Read Return Pending Fifo, the TID of the “dropped read” is allocated into the Shootdown Forward Path Transfer Fifo which synchronizes over to the CCLK domain. The head entry will deallocate, cause the assertion of ddr_coh_RdShotDown_c2a, and drive the shotdown TID onto ddr_coh_DataTID_c2a (this is done during a cycle where ddr_coh_DataValid_c2a will not assert (DDI knows a cycle adhead of time before data will return)).

In the return path, when read data returns the head entry of the Read Return Pending Fifo provides the TID to index into m_RdShdVec_c4a[19:0]. If the corresponding bit set, then ddr_coh_RdShotDown_c2a will assert while ddr_coh_DataValid_c2a is forced low and the shotdown TID is driven onto ddr_coh_DataTID_c2a.

8.4.4 Data Path

Write Data arrives at DDI piped into 4 consecutive144-bit chunks (128b data + 16 ECC bits). The first 144-bits arrives coincident with the write valid signal, TID, HalfMask, and destination address. When the write request arrives the address is checked to make sure it is not outside the range of the memory defined by the DDR configuration registers. The write data is stored in a register file, indexed by the WrTID. The register file is written

in the CCLK domain when the request is issed from COH, and is read out on the DCLK domain when the DDC requests the data for a write request which was previously accepted. The delay between with the register file is written and the earliest DDC can request the write data is guaranteed to be long enough to avoid a synchronization violation on the register file. When data is read out of the register file it is sent to DDC in 4 consecutive 144-bit chunks.

The details of the read datapath are discussed below in the DDD section and DDP unit descriptions.

8.4.5 Requests to non-existent memory

Request to non-existent memory are accesses which have upper address bits set which are outside of the range for the selected DRAM configuration. The CSR DdrxDdiMifCfg1_MemAddrSize[2:0] is used to determine if a request is to non-existent memory. Based on this CSR, the upper bits or the address presented to DDC are forced low (forces address aliasing). The memory requests will complete as normal using the aliased address (i.e. writes to non-existent memory are software errors which will result in data corruption).

It is required that DdrxDdiMifCfg1_MemAddrSize[2:0] be set correctly, otherwise a read to non-existent memory could case fatal errors in the read return logic by resulting in a read which does not get a response from memory (i.e. it maps to a chip select for a non-existent rank). This would throw off the fifo pointers in the read return logic causing reads to return data that was meant to correspond to subsequent reads.

8.4.6 Powerdown

The memory interface includes logic to issue power-down commands to memory if the interface is idle for a user controller number of cycles. Using power-down reduces the power dissipation in the memory DIMMs. It is expected that enabling power-down will have a minimal impact on performance, since wake up from powerdown is on the order of a few cycles. Any impact can be mitigated by increasing the number of idles required before power-down is entered. It may be possible for power-down to impact performance for some code patterns.

8.4.7 Read Time-Out

The DDR unit includes read time-out detection logic which is intended as a debug tool for improperly configured systems (for example if the settings of the DLL in the PHY are incorrectly programmed potentially causing the return of read data to be dropped). The read time-out logic can be used to indicate such a problem. It is not intended for use during normal system operation. It will not precisely indicated which particular read has hung, and it may fire after allowing numerous returns of bad read data in a poorly configured system. The reason for this is that reads return in order. Thus if a particular read is dropped, any subsequent read returning will be applied to the wrong requester. Thus it is only after reads have stoppped returning, that we can be sure that there is a problem when DDR still has one or more reads waiting for data.

In general, we can bound the amount of time that a read should be outstanding once it has been issued to DDC (the Northwest logic memory controller). Since the read-time out logic is never expect to be needed during normal operation the count was chosen to much larger than necessary to be conservative. The count used is 4096 clock cycles (which is probably 8 times the real worst case).

Each of the 28 TIDs have an associated counter which can count to 4096 dclk cycles. When DdrxDdiRdTime-Out_Enable is set, these counters are enabled to start counting when a read of the corresponding TID is issued from DDI to DDC. DdrxDdiRdTimeOut_Enable will be set to a 1 if a read hang is dectected for any read TID (this is sticky and will remain set until it is cleared via the SCB bus (note it is W1C 'write one to clear', so software must write a 1 to the corresponding bit place in order to clear it out)).

If DdrxDdiRdTimeOut_AutoCompletion is set, then if a read is determined to have hung, the DDR unit will return a fake completion message (assertion of ddr_coh_DatValid_c2a or ddr_coh_RdShotDown_c2a). The DDR unit will return whatever data values are in its read data path flops. Note that if the read data was corrupted it may result in an uncorrectable ECC error pattern on the returning data. Read Time-Out AutoCompletion is a feature which is intended to be used primarily for the calibration of the read path DLLs and for debugging, however it can be enabled during normal operation if software finds it useful.

8.4.8 Registers and Definitions

This subsection defines the CSR registers, while the next subsection creates the two instances. The CSRs live in the DdrDdiCsr sub-module of DDI, which runs on the DCLK. The CSRs are written and read via the ICE9

Serial Configuration Bus.

The values of the registers R_DdrxDdcMemCfg1-5, R_DdrxDdcDIMMODT, R_DdrxDdpODT, and R_DdrxDIMMSize may only change prior to the de-assertion of R_DdrxDdcDdpSoftReset. More specific, information is located in the "Reset and Initialization" section of this chapter.

The values of the registers R_DdrxDdiMifCfg1-2 can be changed at any time.

The "SPD Byte #" column in the tables below is provided as a hint as to what information may need to be read from the DIMMs' SPD in order to figure out what value to set for the corresponding CSR field. Note that many of the parameters accessed from SPD are in time units while the many of the corresponding CSRs are in units of DCLK cycles.

8.4.8.1 R_DdrxDdcDdpSoftReset - Soft Reset for DDC and DDP

Register
R_DdrxDdcDdpSoftReset

Address

0x0_0000_0000 (plus base address)

Bit	Mnemonic	Access	Reset	Type	(Product)	Definition
31:3						Reserved
2	InitDimm	RW	0		ICE9B+	1 -> 0 transisiton tells controller to re-issue the initialization sequence to the DIMM. The controller will always issue the initialization sequence after SoftResetDDC is deasserted (goes low) regardless of the state of this InitDimm. InitDimm can be left low if run-time re-initialization is not required.
1	SoftResetDDP	RW	1		ICE9B+	Used as the reset signal for DDP. Separating this from the reset to DDC allows DDP to wake up first and calibrate it's IO driver output impedence, before we wake up DDC and have it start the JEDEC DRAM init sequence
0	SoftResetDDC	RW	1		ICE9B+	Used as the reset signal for DDC. Can only be deasserted after setting the correct CSR values to R_DdrxDdcMemCfg1-5, R_DdrxDdcDIMMODT, R_DdrxDdpODT, and R_DdrxDIMMSize. The de-assertion (transition from HIGH to LOW) causes the DDR2-SDRAM controller to issue the JEDEC standard initialization sequence to the SDRAM devices. (Note the Type "L" is an indication that this is intended to normally be the last CSR written). Overlaps SoftReset.
0	SoftReset	RW	1		ICE9A	Used as the reset signal for DDC and DDP. Can only be deasserted after setting the correct CSR values to R_DdrxDdcMemCfg1-5, R_DdrxDdcDIMMODT, R_DdrxDdpODT, and R_DdrxDIMMSize. The de-assertion (transition from HIGH to LOW) causes the DDR2-SDRAM controller to issue the JEDEC standard initialization sequence to the SDRAM devices. (Note the Type "L" is an indication that this is intended to normally be the last CSR written.

8.4.8.2 R_DdrxDdcMemCfg1 - Memory Controller Configuration Register 1

Register

R_DdrxDdcMemCfg1

Address

0x0_0000_0004 (plus base address)

Bit	Mnemonic	Access	Reset	(Valid Values)	(SPD Byte #)	Definition
31	PchPowerDown	RW	1	0-1		*** This feature in NOT supported. It is a requirement that software write a "0" to PchPowerDown before bringing the DDR interface out of reset. ***
30:26	RAS	RW	0	4-18	30	Active to precharge (tRAS), specified in DCLK cycles
25:23	RCD	RW	0	2-6	29	Active to read or write delay (tRCD), specified in DCLK cycles
22:20	RRD	RW	0	2-4	28	Active bank a to active bank b (tRRD), specified in DCLK cycles
19:17	RP	RW	0	1-6	27	Precharge command period (tRP), specified in DCLK cycles
16:12	RC	RW	0	5-24	41, 40	Active to active/auto-refresh period (tRC), specified in DCLK cycles
11:4	RFC	RW	0	6-255	42, 40	Auto-refresh to active/auto-refresh period (tRFC), specified in DCLK cycles
3:2	RTP	RW	0	2-3	38	Read to precharge delay (tRTP) specified in DCLK cycles
1:0						Reserved

8.4.8.3 R_DdrxDdcMemCfg2 - Memory Controller Configuration Register 2

Register

R_DdrxDdcMemCfg2

Address

0x0_0000_0008 (plus base address)

Bit	Mnemonic	Access	Reset	(Valid Values)	(SPD Byte #)	Definition
31:29	MRD	RW	2	1-7		load mode register cmd to active or refresh, specified in DCLK cycles. 2 is valid minimum value for tMRD for a wide range of DDR2 parts.
28:21						Reserved
20:18	CL	RW	0	4-6	18, (9, 23,25) or sys config file	CAS latency, specified in DCLK cycles (Note: CAS latency of 3 is NOT supported)
17:15	WR	RW	0	2-6	36	Write recovery time (tWR), specified in DCLK cycles
14:12	WTR	RW	0	2-4	37	Write to read cmd delay (tWTR), specified in DCLK cycles
11:9	AL	RW	0	0-5		Additive latency, specified in DCLK cycles Note that non-zero AL values may improve DDR2 bus utilization and hence performance, especially for random access patterns and/or if reads and writes are issued with auto-precharege.
8:4	FAW	RW	14	7-20		Four bank activate period (tFAW), specified in DCLK cycles This defaults to an acceptable value. Other choices are provided below. From JEDEC Spec 79-2B DDR2 400/800 - 35ns => 14 cycles DDR2 333/667 - 37.5ns => 13 cycles DDR2 266/533 - 50ns => 14 cycles
3:0						Reserved

8.4.8.4 R_DdrxDdcMemCfg3 - Memory Controller Configuration Register 3

Register

R_DdrxDdcMemCfg3

Address

0x0_0000_000c (plus base address)

Bit	Mnemonic	Access	Reset	(Valid Values)	(SPD Byte #)	Definition
31:29						Reserved
28	Bankbits	RW	1	0-1	17	Number of bits in the bank address (encoded). Values are mapped as follows: 0 - 2 bank bits (i.e. 4 bank chips) 1 - 3 bank bits (i.e. 8 bank chips)
27:25	Rowbits	RW	0	3-5	3	Number of bits in the row address (encoded) 3 - 14 row bits 4 - 15 row bits 5 - 16 row bits
24:8	Delay	RW	0	10-131071		reset to SDRAM init delay specified in DCLK cycles. Valid values: 10 - 131071 At 400Mhz DDR Delay = 80000 * 2.5ns = 200us (JEDEC requires minimum of 200us)
7:0						Reserved

8.4.8.5 R_DdrxDdcMemCfg4 - Memory Controller Configuration Register 4

Register

R_DdrxDdcMemCfg4

Address

0x0_0000_0010 (plus base address)

Bit	Mnemonic	Access	Reset	(Valid Values)	(SPD Byte #)	Definition
31:16	REFI	RW	0	10-65535	12	Period between auto-refresh commands issued by the controller, specified in DCLK cycles. ref = auto refresh interval/tCK tREFI should be set to 7.8us. 400MHz => 3125 333MHz => 2604 267MHz => 2083 Note: JEDEC 79-2B requires setting tREFI to 3.9us if 85 degrees C < tCASE <= 95 degrees C. Preliminary studies show that tCASE is expected to be below 70 degrees in our system.
15	Regdimm	RW	0	0-1		Set when using registered / buffered DIMM.
14	DS	RW	0	0-1	22	DDR2 drive strength setting programmed into Extended Mode Register Bit 1. Values mapped to EMR as follows (refer to DDR2 SDRAM device data-sheet for description of drive strength settings): 0 - EMR[1] = 0 1- EMR[1] = 1 (SPD Byte #22 reports whether this is supported)
13:12	Rtt	RW	2	0-3	22	ODT effective resistance Rtt. DDR2 On-Die Termination effective resistance setting programmed into Extended Mode Register bits 2 and 6. Values mapped to EMR as follows: 0 - EMR[6] = 0, EMR[2] = 0 (Rtt disabled) 1 - EMR[6] = 0, EMR[2] = 1 (75 ohms) 2 - EMR[6] = 1, EMR[2] = 0 (150 ohms) 3 - EMR[6] = 1, EMR[2] = 1 (50 ohms (not supported on slower memory)) SPD Byte #22 reports whether 50 ohms is supported 150 ohm setting may be appropriate for interfacing to 1 and 2 rank DDR2 DIMMs running at 333/667 or 400/800.
11	Qoff	RW	0	0-1		SDRAM output enable function. This signal is passed to bit E12 of the Extended Mode Register during initialization. Typically set to '0' to enable data and strobe outputs from the SDRAM devices. Can be set to '1' for IDD characterization of read current.
10:0						Reserved

8.4.8.6 R_DdrxDdcMemCfg5 - Memory Controller Configuration Register 5

Register

R_DdrxDdcMemCfg5

Address

0x0_0000_0014 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:16	emr2	RW	0	Value programmed into DIMM's Extended Mode Register 2 during initialization. Most DDR2 SDRAM devices specify all of these bits as reserved (must be set to 0).
15:0	emr3	RW	0	Value programmed into DIMM's Extended Mode Register 3 during initialization. Most DDR2 SDRAM devices specify all of these bits as reserved (must be set to 0).

8.4.8.7 R_DdrxDdcMemCfg6 - Memory Controller Configuration Register 6

Register

R_DdrxDdcMemCfg6

Address

0x0_0000_0018 (plus base address)

Bit	Mnemonic	Access	Reset	(Valid Values)	Definition
31:18					Reserved
17	IdleBusDrive	RW	1	0-1	Causes DQ and DQS to be driven during idle periods (when no read nor writes are occuring). If this bit is set, the bus will be driven during idle periods as follows: - After a write, bus will remain driven. DQ lines will be driven with value of last data phase. - After a read, bus will be driven # clocks after the end of the read postamble where # is selected using ReadToIdleDriveDelay. The bus will be driven to a value of 72'haa_aaaa_aaaa_aaaa_aaaa.
16:15	ReadToIdleDriveDelay	RW	3	0-3	Delay to DQSP, DQSN, and DQ output enable switch-on after a read command relative to end of read postamble. 0x0 : -1.0 clocks 0x1 : 0 clocks 0x2 : 1.0 clocks 0x3 : 2.5 clocks
14	LookaheadPch	RW	1	0-1	Look ahead precharge enable. When enabled the controller will look ahead into the command queue and analyze the queued requests and perform precharge operations as soon as possible in order to maximize bandwidth efficiency. 0 - disable 1 - enable

13	LookaheadAct	RW	1	0-1	Look ahcad activate enable. When enabled, the controller will look ahead into the command queue and analyze the queued requests and perform activate operations as soon as possible in order to maximize bandwidth efficiency. 0 - disable 1 - enable
12	LookaheadApch	RW	0	0-1	Look ahead auto-precharge enable. When enabled the controller will look ahead into the command queue and analyze the queued requests and perform an auto-precharge operation to the current read or write operation in order to maximize bandwidth efficiency. 0 - disable 1 - enable
11	OdtAdvTurnOn	RW	0	0-1	Advances ODT turn-on by one clock (only supported for cas latencies: CL5, CL6)
10	OdtDelayTurnOff	RW	0	0-1	Delay ODT turn-off by on clock
9	TwoTMode	RW	0	0-1	Two cycle timing (2T) enable. When enabled, the controller extends the timing of the SDRAM control signals (ras, cas, and we) to be two clocks in duration. 1 - enable 0 - disable
8	TwoTModeSelCycle	RW	1	0-1	Two cylce timing cycle select. Controls which phase of the two clock cycle command period the cs_n is asserted. 0 - cs_n asserted during the first cycle 1 - cs_n asserted during the second cycle.
7:6	ReadToWrite	RW	1	1, 2, 3	Read to write delay (valid values: 1,2,3)
5:3	WriteToWrite	RW	1	0-7	Minimum delay from write to write (different ranks). NOTE: that zero is a legal choice ONLY if R_DdrxDdcDIMMODT_OdtWrMapCs* = 0000. (Setting this to zero, can cause ODT problems, as the ODT spec requires turn on 3 cycles before the data and turn off 2 cycles before the data, thus if the data to different ranks was back to back, then switching to the ODT for the second write causes the first to switch prematurely)
2:0	ReadToRead	RW	1	0-7	Minimum delay from read to read (different ranks). NOTE: that zero is a legal choice ONLY if R_DdrxDdcDIMMODT_OdtRdMapCs* = 0000. (Setting this to zero, can cause ODT problems, as the ODT spec requires turn on 3 cycles before the data and turn off 2 cycles before the data, thus if the data from different ranks was back to back, then switching to the ODT for the second read causes the first to switch prematurely). NOTE: also that a value of zero may have the potential of resulting in output drive contention between ranks.

8.4.8.8 R_DdrxDdcMemCfg7 - Memory Controller Configuration Register 7

Register

R_DdrxDdcMemCfg7

Address

0x0_0000_001c (plus base address)

Bit	Mnemonic	Access	Reset	(Valid Values)	Definition
31:23					Reserved
22	InitAutoInitDisable	RW	0	0-1	Disables automatic initialization handled by controller
21:18	InitMr	RW	0		Mode Register to write to
17:2	InitMrData	RW	0		Contents to write to mode register
1	InitPrechargeAll	RW	0	0-1	Issue precharge-all command
0	InitRefresh	RW	0	0-1	Issue refresh command

8.4.8.9 R_DdrxDdcDIMMODT - Memory Controller ODT Selection Matrix Configuration

The defaults for R_DdrxDdcDIMMODT are expected to be appropriate for the target single and dual rank configurations of one DIMM slot based on reviewing preliminary termination matrix recommendations presented by Samsung for 667 data rate operation and Micron for 667 and 800 data rates. We plan to follow the industry recommendations for single-DIMM-slot designs, which call for ODT on the active DIMM rank only, during writes, and ODT on the controller only, during reads.

Register

R_DdrxDdcDIMMODT

Address

0x0_0000_0020 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:28	OdtRdMapCs0	RW	0	Selects which DRAM ODT outputs are enabled when reading from chip select 0. ex: odt_rd_map_cs0=4'b1110 will enable odt[1], odt[1], and odt[2] during a read from memory devices on chip select 0.
27:24	OdtRdMapCs1	RW	0	Selects which DRAM ODT outputs are enabled when reading from chip select 1.
23:20	OdtRdMapCs2	RW	0	Selects which DRAM ODT outputs are enabled when reading from chip select 2.
19:16	OdtRdMapCs3	RW	0	Selects which DRAM ODT outputs are enabled when reading from chip select 3.
15:12	OdtWrMapCs0	RW	1	Selects which DRAM ODT outputs are enabled when writing to chip select 0
11:8	OdtWrMapCs1	RW	2	Selects which DRAM ODT outputs are enabled when writing to chip select 1
7:4	OdtWrMapCs2	RW	0	Selects which DRAM ODT outputs are enabled when writing to chip select 2
3:0	OdtWrMapCs3	RW	0	Selects which DRAM ODT outputs are enabled when writing to chip select 3

8.4.8.10 R_DdrxDdpODT - On-Die-Termination resistance value on ICE9 DDR2-I/O PADs during reads

Register

R_DdrxDdpODT

Address

0x0_0000_0024 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:30	OdtValue	RW	0	On-Die-Termination value used in the DDR PHY IO cells. Maps to the values driven into the {TERM150, TERM300} pins of the ARM IO cell. 00 - Rx Mode, ODT disabled 01 - Rx Mode, 150 ohm calibrated ODT 10 - UNDEFINED IN ARM SPEC 11 - Rx Mode, 75 ohm calibrated ODT The 150 Ohm setting is expected to be sufficient. However, it may necessary to use the 75 Ohm setting for 400/800 systems.
29:0				Reserved

8.4.8.11 R_DdrxDIMMSize - Size of the DIMM this DDR unit instance is interfacing with.

Register

R_DdrxDIMMSize

Attributes

-kernel

Address

0x0_0000_0028 (plus base address)

Bit	Mnemonic	Access	Reset	(SPD Byte #)	Definition
31:3					Reserved
2:0	DIMMSize	RW	0	5, 31	Total memory connect to this DDR interface (half of the total main memory space per ICE9). DIMM Rank Density * Number of Ranks Used to filter out requests to non-existent memory. Valid values 0 - 4 0 - 1GB 1 - 2GB 2 - 4GB 3 - 8GB 4 - 16GB

8.4.8.12 R_DdrxDdiMifCfg1 - Memory Interface Configuration Register 1

Register

R_DdrxDdiMifCfg1

Address

0x0_0000_002c (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:9				Reserved
8:1	ArbPrefWheel	RW	0x0F	Each bit set represents an additional 1 out of 10 cycles where reads have arbitration preference over writes. This allows for performance tuning by allowing more/less reads to pass independent write requests in DDI. Note: 1. ArbPrefWheel should always be programmed with contiguous bits set (to minimize DDR bus turn around time penalty of switching from reads to writes or vice-versa. More specifically, ArbPrefWheel should be programmed to one of the following values: 00000000 00000001 00000011 00000111 00001111 00011111 00111111 01111111 11111111 2. the arbitration preference for 2 out of 10 cycles is not user controllable, but dedicated 1 for read and 1 for writes to prevent starvation if a user sets (or clears) all the bits of ArbPrefWheel.
0	AutoPch	RW	1	The auto-precharge option is useful where the access patterns tend to be random (as seen at the DDR2 interface). With random sequences, banks are rarely left open with the exact row required by a subsequent request. If auto-precharge was not used for the previous access to a bank, subsequent accesses to that bank first require the bank to be closed (prechareged), causing a delay. 0 - Requests issued as read / write without auto-precharge 1 - Requests issued as read / write with auto-precharge

8.4.8.13 R_DdrxDdiMifCfg2 - Memory Interface Configuration Register 2

Register

R_DdrxDdiMifCfg2

Address

0x0_0000_0030 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:19				Reserved
18	PwrDnEnable	RW	1	0 - DDR2 is never issued the power-down command 1 - DDR2 is issued the power-down command if the no read or write requests are sent to the memory interface for a period of time determined by the PwrDnCount setting.
17:0	PwrDnCount	RW	128	Number of ICE9 core clock (cclk) idle cycles before a power-down command is issued to memory. This is required to be set to a value larger than (Twait = 2 * R_DdrxDdcMemCfg1_RFC) in dclks. Examples for DIMMs configured with 1Gb devices: cclk/dclk Twait 250/400 - Twait = 102 dclks, PwrDnCount >= 64 cclks 250/333 - Twait = 86 dclks, PwrDnCount >= 54 cclks 250/267 - Twait = 68 dclks, PwrDnCount >= 43 cclks Note the R_DdrxDdcMemCfg1_RFC value used in these calcualtions are from "Table 39 - Refresh parameters by device density" of JESD79-2B (JEDEC Standard - DDR2 SDRAM Specification).

8.4.8.14 R_DdrxPhyCfg1 - PHY Interface Configuration Register 1

Register

R_DdrxPhyCfg1

Address

0x0_0000_0034 (plus base address)

Bit	Mnemonic	Access	(Valid Values)	Reset	Definition
31:12					Reserved
11:9	DqsOeOn	RW	0 - 5	2	DQS output enable switch-on time relative to start of write preamble. 0x0: -1.5 clocks 0x1: -1.0 clocks 0x2: -0.5 clocks 0x3: 0 clocks
8:6	DqsOeOff	RW	0 - 7	3	DQS output enable switch-off time relative to end of write postamble. 0x0: -1.5 clocks 0x1: -1.0 clocks 0x2: -0.5 clocks 0x3: 0 clocks 0x4: 0.5 clocks 0x5: 1 clocks 0x6: 1.5 clocks 0x7: 2.0 clocks
5:3	DqOeOn	RW	0 - 5	2	DQ output enable switch-on time relative to start of wirte preamble. 0x0: -1.25 clocks 0x1: -0.75 clocks 0x2: -0.25 clocks 0x3: 0.25 clocks
2:0	DqOeOff	RW	0 - 7	2	DQ output enable switch-off time relative to end of write postamble. 0x0: -1.25 clocks 0x1: -0.75 clocks 0x2: -0.25 clocks 0x3: 0.25 clocks 0x4: 0.75 clocks 0x5: 1.25 clocks 0x6: 1.75 clocks 0x7: 2.25 clocks

8.4.8.15 R_DdrxPhyCfg2 - PHY Interface Configuration Register 2

Register

R_DdrxPhyCfg2

Address

0x0_0000_0038 (plus base address)

Bit	Mnemonic	Access	(Valid Values)	Reset	Definition
31:12					Reserved

11:9	AsicDqsOdtOn	RW	0 - 5	2	Note there are two changes going from ICE9A to ICE9B: First - Bugzilla 2401 was fixed. Secondly - the range of adjustability was changed based on feedback from debug lab bringup studies on ice9a parts. DQS resistor output enable (ASIC side ODT) and pad input enable (IE-to-Y) switch-on time relative to start of read preamble. ICE9A RANGE: 0x0: -2.5 clocks (Not supported if AsicDqsOdtOff is set to 0x6 or 0x7 (Bugzilla 2401)) 0x1: -2.0 clocks (Not supported if AsicDqsOdtOff is set to 0x6 or 0x7 (Bugzilla 2401)) 0x2: -1.5 clocks 0x3: -1.0 clocks 0x4: -0.5 clocks 0x5: 0 clocks ICE9B+ RANGE: 0x0: -1.5 clocks 0x1: -1.0 clocks 0x2: -0.5 clocks 0x3: 0 clocks 0x4: 0.5 clocks 0x5: 1.0 clocks 0x6: 1.5 clocks 0x7: 2.0 clocks Note: The ARM SSTL18 output buffer contains an AND gate which will disable the output enable when the resistor output enable is switched on.

8:6	AsicDqsOdtOff	RW	0 - 7	3	Note there are two changes going from ICE9A to ICE9B: First - Bugzilla 2401 was fixed. Secondly - the range of adjustability was changed based on feedback from debug lab bringup studies on ice9a parts. DQS resistor output enable (ASIC side ODT) and pad input enable (IE-to-Y) switch off time relative to the end of read postamble. ICE9A RANGE: 0x0: -1.5 clocks 0x1: -1.0 clocks 0x2: -0.5 clocks 0x3: 0 clocks 0x4: 0.5 clocks 0x5: 1.0 clocks 0x6: 1.5 clocks (Not supported if AsicDqsOdtOn is set to 0x0 or 0x1 (Bugzilla 2401)) 0x7: 2.0 clocks (Not supported if AsicDqsOdtOn is set to 0x0 or 0x1 (Bugzilla 2401)) ICE9B+ RANGE: 0x0: -0.5 clocks 0x1: 0 clocks 0x2: 0.5 clocks 0x3: 1.0 clocks 0x4: 1.5 clocks 0x5: 2.0 clocks 0x6: 2.5 clocks 0x7: 3.0 clocks Note: The output enable of the ARM SSTL18 I/O buffer will be disabled as long as the resistor output enable (ROE) pin is asserted. Care must be taken to ensure that longer ROE switch off times do not interfere with subsequent writes. The timing of subsequent writes can be contolled using R_DdrxDdcMemCfg6_ReadToWrite

5:3	AsicDqOdtOn	RW	0 - 5	1	Note there are two changes going from ICE9A to ICE9B: First - Bugzilla 2401 was fixed. Secondly - the range of adjustability was changed based on feedback from debug lab bringup studies on ice9a parts. DQ resistor output enable (ASIC side ODT) and pad input enable (IE-to-Y) switch-on time relative to start of read preamble. ICE9A RANGE: 0x0: -2.5 clocks (Not supported if AsicDqsOdtOff is set to 0x6 or 0x7 (Bugzilla 2401)) 0x1: -2.0 clocks (Not supported if AsicDqsOdtOff is set to 0x6 or 0x7 (Bugzilla 2401)) 0x2: -1.5 clocks 0x3: -1.0 clocks 0x4: -0.5 clocks 0x5: 0 clocks ICE9B+ RANGE: 0x0: -1.5 clocks 0x1: -1.0 clocks 0x2: -0.5 clocks 0x3: 0 clocks 0x4: 0.5 clocks 0x5: 1.0 clocks 0x6: 1.5 clocks 0x7: 2.0 clocks Note: The ARM SSTL18 output buffer contains an AND gate which will disable the output enable when the resistor output enable is switched on.

2:0	AsicDqOdtOff	RW	0 - 7	3	Note there are two changes going from ICE9A to ICE9B: First - Bugzilla 2401 was fixed. Secondly - the range of adjustability was changed based on feedback from debug lab bringup studies on ice9a parts. DQ resistor output enable (ASIC side ODT) and pad input enable (IE-to-Y) switch off time relative to the end of read postamble. ICE9A RANGE: 0x0: -1.5 clocks 0x1: -1.0 clocks 0x2: -0.5 clocks 0x3: 0 clocks 0x4: 0.5 clocks 0x5: 1.0 clocks 0x6: 1.5 clocks (Not supported if AsicDqsOdtOn is set to 0x0 or 0x1 (Bugzilla 2401)) 0x7: 2.0 clocks (Not supported if AsicDqsOdtOn is set to 0x0 or 0x1 (Bugzilla 2401)) ICE9B+ RANGE: 0x0: -0.5 clocks 0x1: 0 clocks 0x2: 0.5 clocks 0x3: 1.0 clocks 0x4: 1.5 clocks 0x5: 2.0 clocks 0x6: 2.5 clocks 0x7: 3.0 clocks Note: The output enable of the ARM SSTL18 I/O buffer will be disabled as long as the resistor output enable (ROE) pin is asserted. Care must be taken to ensure that longer ROE switch off times do not interfere with subsequent writes. The timing of subsequent writes can be contolled using R_DdrxDdcMemCfg6_ReadToWrite

8.4.8.16 R_DdrxPhyCfg3 - PHY Interface Configuration Register 3

Register

R_DdrxPhyCfg3

Address

0x0_0000_003c (plus base address)

Bit	Mnemonic	Access	(Valid Values)	Reset	Definition
31:14					Reserved
13:11	DqsPreambleEnnOn	RW	0 - 5	2	Read preamble enable switch-on time relative to start of read preamble. 0x0: -0.5 clocks 0x1: 0 clocks 0x2: 0.5 clocks 0x3: 1.0 clocks 0x4: 1.5 clocks 0x5: 2.0 clocks
10:8	DqsPreambleEnnOff	RW	0 - 7	2	Read preamble enable switch-off time relative to the third edge of the read DQS. 0x0: -1.0 clocks 0x1: -0.5 clocks 0x2: 0 clocks 0x3: 0.5 clocks 0x4: 1.0 clocks 0x5: 1.5 clocks 0x6: 2.0 clocks 0x7: 2.5 clocks
7:0					Reserved

8.4.8.17 R_DdrxDdpDLLLane0 - PHY Read Lane 0 DLL Configuration Register

Register

R_DdrxDdpDLLLane0

Address

0x0_0000_0040 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.18 R_DdrxDdpDLLLane1 - PHY Read Lane 1 DLL Configuration Register

Register

R_DdrxDdpDLLLane1

Address

0x0_0000_0044 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.19 R_DdrxDdpDLLLane2 - PHY Read Lane 2 DLL Configuration Register

Register

R_DdrxDdpDLLLane2

Address

0x0_0000_0048 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.20 R_DdrxDdpDLLLane3 - PHY Read Lane 3 DLL Configuration Register

Register

R_DdrxDdpDLLLane3

Address

0x0_0000_004c (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.21 R_DdrxDdpDLLLane4 - PHY Read Lane 4 DLL Configuration Register

Register

R_DdrxDdpDLLLane4

Address

0x0_0000_0050 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.22 R_DdrxDdpDLLLane5 - PHY Read Lane 5 DLL Configuration Register

Register

R_DdrxDdpDLLLane5

Address

0x0_0000_0054 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.23 R_DdrxDdpDLLLane6 - PHY Read Lane 6 DLL Configuration Register

Register

R_DdrxDdpDLLLane6

Address

0x0_0000_0058 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.24 R_DdrxDdpDLLLane7 - PHY Read Lane 7 DLL Configuration Register

Register

R_DdrxDdpDLLLane7

Address

0x0_0000_005c (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.25 R_DdrxDdpDLLLane8 - PHY Read Lane 8 DLL Configuration Register

Register

R_DdrxDdpDLLLane8

Address

0x0_0000_0060 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24				Reserved
23:16	MasterAdj	RW	186	Master Delay Adjustment - specifies the number of slave adjustment steps. (See DLL description of DDP Unit for details settings based on clock frequency).
15:8	Slave0Adj	RW	1	Slave DLL to delay dummy DQS to match the DQS board trace delay to and from DIMM. (See DLL description of DDP Unit for details on settings).
7:0	Slave1Adj	RW	12	Slave DLL to delay DQS nomially by 1/4 DCLK. (See DLL description of DDP Unit for details settings based on clock frequency).

8.4.8.26 R_DdrxDdpDLLReset - PHY DLL Reset

Register

R_DdrxDdpDLLReset

Address

0x0_0000_0064 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:1				Reserved
0	Reset	RW	1	Active high reset routed to each of the DLLs in the PHY. Direct access is provided for the DLL reset since the TrueCircuits documentation says that DLL fault testing should be done with the DLL reset asserted.

8.4.8.27 R_DdrxDdpCKReset - Reset for CK clock outputs to DIMM

Register

R_DdrxDdpCKReset

Address

0x0_0000_0164 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:1				Reserved
0	Reset	RW	1	Deasserting this CSR bit causes the PHY to start driving clocks to the DIMMs. Before deasserting this bit, software must make sure dclk and dmclk90 are stable and that ClkDrivImped of R_DdrxDdpCmdDrv is set to an appropriate value.

8.4.8.28 R_DdrxDddRdDelay

Register

R_DdrxDddRdDelay

Address

0x0_0000_0068 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:1				Reserved
0	DelayFifoReadOut	RW	0	Setting this to a 1'b1 adds an extra cclk cycle of latency to the read return path as a debug mechanism to prove bugs are not due to read return fifo underflow.

8.4.8.29 R_DdrxDdiMemLoopBack

Register

R_DdrxDdiMemLoopBack

Address

0x0_0000_006c (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:1				Reserved
0	MemLoopBack	RW	1	When this set to "1" read and write requests received by DDR will receive a fake completion response (i.e. will not really issue to memory and will return meaningless data. This is only expected to be used during the initial boot sequence where it is possible for the reads and writes will show up at the DDR unit, that don't need complete correctly. This is because of the boot sequence involves the boot processor doing writes to the cache which will result in the caches doing reads for allocation before allowing the write (which it thinks is necessary for coherance). This CSR bit needs to be cleared before R_DdrxDdcDdpSoftReset is de-asserted and MemLoopBack must never be asserted when R_DdrxDdcDdpSoftReset is deasserted.

8.4.8.30 R_DdrxDdiRdPathRst

Register

R_DdrxDdiRdPathRst

Address

0x0_0000_0070 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:1				Reserved
0	RdPathRst	RW	0	This is NOT intended for general use. It is a hook for debugging potential issues with the PHY DLL settings. When asserted state elements in the read return datapath are forced to their reset values. A read can not be outstanding when this is asserted, this must be deasserted before any read is issued to the DDR unit. When this CSR changes value, it is NOT allowed to change value again from at least 10 dclk cycles (Note: that this require should be meet by default since it takes at least 30 clock cycles to affect the same CSR with back to back SCB writes to it).

8.4.8.31 R_DdrxDdiRdTimeOut

Register

R_DdrxDdiRdTimeOut

Attributes

-writeonemixed

Address

0x0_0000_0074 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:3				Reserved
2	Enable	RW	0	Enable the counters which are used to determine if a read hangs.
1	AutoComplete	RW	0	Causes the DDR unit to issue a false read completion for reads the hang. See description of Read Time-Out AutoCompletion in the DDI subsection of this spec.
0	RdHang	RW1C	0	Set if a read has timed out. The value is sticky until software writes a 1 to clear it.

8.4.8.32 R_DdrxDdpCalReset

Register

R_DdrxDdpCalReset

Address

0x0_0000_0078 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:1				Reserved
0	CalReset	RW	1	When asserted, the calibratin logic in the DDR2-PHY will be held in reset. After deasserting the "dclk and cclk resets" which go to DDI, R_DdrxDdpImpedCal_CalClk should be set, then CalReset can be deasserted.

8.4.8.33 R_DdrxDdpCalError

Register

R_DdrxDdpCalError

Attributes

-writeonemixed

Address

0x0_0000_007c (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:20				Reserved
19	CalUpdate	RW1C	0	Set when the calibration logic updates ImpP and ImpN
18	CalErrDerate	RW	1	If the auto-cal logic very rarely asserts cal_fault_occur or cal_timout_occur, there may not be a problem. CalErrDerate allows users to cause the decrementing of CalErrCount every time the auto-cal logic runs for 524,288 cycles without a cal_fault or a cal timeout.
17	CalErrInterrupt	RW1C	0	Asserted when CalErrCount has reached the IntReportThreshold. This bit is sticky until software does a write one to clear it.
16:9	CalErrCount	R	0	8 bit saturating counter. Increments when ever the auto-calibration logic in the DDR-PHY asserts cal_fault_occur or cal_timout_occur. NOTE that CalErrCount automatically cleared whenever CalErrInterrupt is cleared.
8:1	IntReportThreshold	RW	5	Asserts an interrupt if CalErrCount goes above this specified value. (Valid values 1-255).
0	CalErrIntEnable	RW	1	Settng this bit enables interrupts to be reported for auto-calibration errors based on the settings of the other fields of this CSR. Setting this to zero forces both CalErrCount and CalErrCount to be zero.

8.4.8.34 R_DdrxDdpCalEnable

Register

R_DdrxDdpCalEnable

Address

0x0_0000_0080 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:1				Reserved
0	CalEnable	RW	0	If CalEnable is low when DdcDdpSoftReset is deasserted, the the intial calibration settings updated into the IOs will be the worst case SS corner setting (relatively strong calibration settings) (impP=12, impN=9). If CalEnable is never asserted, these values will be permanently used. Once CalEnable has been asserted, calibration values will be updated into the IOs according the settings of the other Cal related CSRs. If CalEnable is deassertd at some point, the values of R_DdrxDdpImpedCal_LastUpdatedImpP/N. It is recommended that users not toggle CalEnable, but choose whether to leave it asserted or deasserted, and uses the finer grain controls of the DdrxDdpImpedCal register to control update frequency and temporary disabling.

8.4.8.35 R_DdrxDdpCalCounter

Register

R_DdrxDdpCalCounter

Address

0x0_0000_0084 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:16				Reserved
15:0	CalCounter	RW	0	Determines the period between IO calibration updates if AutoCalUpdate is enabled. CalCounter is the upper 16 bits of a 32 bit count down counter, thus it decrements once every 65536 dclk cycles, thus a value of 1 means do an IO cal update once every 65536 dclk cycles. Setting this to zero means to do a cal update on the first opportunity after the calibrator has come up with a new value. When counter reaches zero it means to update the IO calibration on the next opportunity according to CalMode and OverrideAutoCalibrtion.

8.4.8.36 R_DdrxDdpImpedCal

Register

R_DdrxDdpImpedCal

Address

0x0_0000_0088 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31				Reserved

30	ManualCalUpdate0to1	RW	0	0->1 transition tells DDI to update the IOs with calibration values based on CalMode and OverrideAutoCalibration at the next opportunity. This bit should not be used when AutoCalUpdate is set. They are mutually exclusive ways of controlling calibration value updates.
29	AutoCalUpdate	RW	0	1 - DDI will update the IOs with calibration values based on CalMode and OverrideAutoCalibration at the next opportunity after CalCounter counts down to zero. 0 - Software must specifically initiate calibration value updates with ManualCalUpdate0to1
28:27	CalClk	RW	0	0 - CalClk = dclk/2 1 - CalClk = dclk/4 2 - CalClk = dclk/8 3 - CalClk = dclk/16 (Note: 1. R_DdrxDdpCalReset must be asserted when changing the value of CalClk. 2. CalClk is required to be less than 300MHz)
26:25	CalMode	RW	0	See decision of IO calibration from more information on the CalModes: 0 - update IO calibration during DIMM auto refresh operation. 1 - update IO calibration during DIMM refresh operation, while zeroing the dram clk for one cycle 2 - update IO calibration during precharge powerdown, while zeroing the dram clk for one cycle. Note that Cal Mode 2 requires R_DdrxDdiMifCfg2_PwrDnEnable to be set to 1 (otherwise the logic may hang waiting for a powerdown event which will never happen, and thus block forward progress for memory requests). This mode also requires R_DdrxDdcMemCfg1_PchPowerDown to be set to 1. 3 - update IO calibration during dram self-refresh. CalModes 2 and 3 may have a noticable impact on performance if the CalCounter is set to zero or a small value.
24	OverrideAutoCalibration	RW	0	Override the auto calibration values computed and instead update the I/O pads with the values of OverrideImpP and OverrideImpN provided by this CSR.
23:20	OverrideImpP	RW	0	User supplied value for pull-up impedence calibration.
19:16	OverrideImpN	RW	0	User supplied value for pull-down impedence calibration.

15:12	LastUpdatedImpP	R	0	The current calibration value loaded into the register which drives ImpP to the level shifter in the IO ring.
11:8	LastUpdatedImpN	R	0	The current calibration value loaded into the register which drives ImpN to the level shifter in the IO ring.
7:4	ImpP	R	0	Value determined by auto calibration logic which currently needs to be feed into the IO pads to adjust the pull-up impedence for outputs and input termination.
3:0	ImpN	R	0	Value determined by auto calibration logic which currently needs to be feed into the IO pads to adjust the pull-down impedence for outputs and input termination.

Note that **CalMode 2** is currently unsupported in general use. See bugzilla 2013, quoted here:

> When setting AutoCalUpdate in cal mode 2 (update during prechargePowerdown) the Ddi can hang. This is caused when a request is at the head of the queue requesting to be sent to the controller at the time we start the calibration update process. The calibration logic spins in place waiting for powerdown entry. However, this pending request causes the powerdown counter to be cleared on every cycle, which blocks the Ddr from ever entering powerdown mode.

If CalMode 2 is used, provision must be made to either ensure that no memory references are outstanding at the time that a calibration cycle is initiated, or that some processor is capable of unjamming the autocal sequencer. If you don't understand this, then note that CalMode 2 is currently unsupported.

8.4.8.37 R_DdrxDdpDataDrv

Register

R_DdrxDdpDataDrv

Address

0x0_0000_008c (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:27				Reserved
26:24	DqBl8DrivImped	RW	5	Byte Lane 8 Output Driver Strength 111 - UNDEFINED 110 - UNDEFINED 101 - Tx Mode 60 Ohm (4.7mA) 100 - Tx Mode 40 Ohm (7.0mA) 011 - Tx Mode 24 Ohm (11.7mA) 010 - Tx Mode 20 Ohm (14.0mA) 001 - UNDEFINED 000 - Tx Mode 17 Ohm (16.5mA)
23:21	DqBl7DrivImped	RW	5	Byte Lane 7 Output Driver Strength
20:18	DqBl6DrivImped	RW	5	Byte Lane 6 Output Driver Strength
17:15	DqBl5DrivImped	RW	5	Byte Lane 5 Output Driver Strength
14:12	DqBl4DrivImped	RW	5	Byte Lane 4 Output Driver Strength
11:9	DqBl3DrivImped	RW	5	Byte Lane 3 Output Driver Strength
8:6	DqBl2DrivImped	RW	5	Byte Lane 2 Output Driver Strength
5:3	DqBl1DrivImped	RW	5	Byte Lane 1 Output Driver Strength
2:0	DqBl0DrivImped	RW	5	Byte Lane 0 Output Driver Strength

8.4.8.38 R_DdrxDdpDQSDrv

Register

R_DdrxDdpDQSDrv

Address

0x0_0000_0090 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:27				Reserved
26:24	Dqs8DrivImped	RW	5	DQS8 Output Driver Strength 111 - UNDEFINED 110 - UNDEFINED 101 - Tx Mode 60 Ohm (4.7mA) 100 - Tx Mode 40 Ohm (7.0mA) 011 - Tx Mode 24 Ohm (11.7mA) 010 - Tx Mode 20 Ohm (14.0mA) 001 - UNDEFINED 000 - Tx Mode 17 Ohm (16.5mA)
23:21	Dqs7DrivImped	RW	5	DQS7 Output Driver Strength
20:18	Dqs6DrivImped	RW	5	DQS6 Output Driver Strength
17:15	Dqs5DrivImped	RW	5	DQS5 Output Driver Strength
14:12	Dqs4DrivImped	RW	5	DQS4 Output Driver Strength
11:9	Dqs3DrivImped	RW	5	DQS3 Output Driver Strength
8:6	Dqs2DrivImped	RW	5	DQS2 Output Driver Strength
5:3	Dqs1DrivImped	RW	5	DQS1 Output Driver Strength
2:0	Dqs0DrivImped	RW	5	DQS0 Output Driver Strength

8.4.8.39 R_DdrxDdpCmdDrv

Register

R_DdrxDdpCmdDrv

Address

0x0_0000_0094 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:9				Reserved
8:6	AddrDrivImped	RW	5	Output Driver Strength for address/command (A[15:0], BA[2:0], RAS, CAS, WE) 111 - UNDEFINED 110 - UNDEFINED 101 - Tx Mode 60 Ohm (4.7mA) 100 - Tx Mode 40 Ohm (7.0mA) 011 - Tx Mode 24 Ohm (11.7mA) 010 - Tx Mode 20 Ohm (14.0mA) 001 - UNDEFINED 000 - Tx Mode 17 Ohm (16.5mA)
5:3	CntrDrivImped	RW	5	Output Driver Strength for ODT, CKE, CS
2:0	ClkDrivImped	RW	5	Output Driver Strength for CK

8.4.8.40 R_DdrxDdiPHYWrptrCopy - This read only CSR is intended to be used for debugging only. The values only become valid after the last outstanding read has completed. The pointer is gray coded. When all outstanding reads have completed, the value of the R_DdrxDdiPHYWrptrCopy is expected to be 0001, 0111, 1101, or 1011.

Register

R_DdrxDdiPHYWrptrCopy

Address

0x0_0000_0098 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:4				Reserved
3:0	PHYWrptrCopy	R	1	Copy of the PHY's fifo wr pointer. Value only valid when NO reads are outstanding.

8.4.8.41 R_DdrxDdpHoldFix - This register has be included as a preventive measure. If it turns out that there are hold time problems with the sending of cmd/addr signals to the DIMM. Setting bits in this register muxes in delay elements to add additional hold time margin.

Register

R_DdrxDdpHoldFix

Address

0x0_0000_009c (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:4				Reserved
3	DelayCsn	RW	0	Adds delay to chip selects
2	DelayOdt	RW	0	Adds delay to odt signals
1	DelayCke	RW	0	Adds delay to CKE
0	DelayAddr	RW	0	Adds delay to address signals

8.4.8.42 R_DdrxDdpHighSpeedTest - This CSR is only intended for use during chip testing, where a tester is acting as a DIMM.

Register

R_DdrxDdpHighSpeedTest

Address

0x0_0000_00a0 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:2				Reserved
1	WrTestFakeWrites	RW	0	Causes DDI to: 1. Hold the wr_req line high so that it is constantly issuing write request to the the NWL memory controller (DDC). Each write request uses a randomly generated address. Note the address spans the full 16GB logical address space. 2. Whenever the NWL logic controller gives a write data grant, DDI will send in data a data pattern to the NWL logic controller such that the even DQ bits will toggle for the first for four DQS clock edges, and then the odd DQ bits will toggle for the last four DQS edges of the transfer to the DIMM. The DM bits will toggle every other DQS clock edge.
0	RdTestDQSDLLBypass	RW	0	Setting this HIGH forces a HIGH onto the dll_bypass_slave inputs to the DDR-PHY byte lanes. This is needed during high speed read testing of the DDR PHY so that the tester can drive a pre-shifted DQS (relative to the DQ) and directly write data into the DDR-PHY's read fifo. NOTE: This toggles logic which crosses between two clock domains, thus all logic should be quieted for a few cycles before and after this signal is written. To meet this requirement the following is required: Tests that change the value of the CSR are required to first issue a read to this CSR, folllowed by the write to this CSR and then followed by another read to this CSR. No other action is allowed until the second read of the written data comes back.

8.4.8.43 R_DdrxDdiECCCaptureEnable

Register

R_DdrxDdiECCCaptureEnable

Address

0x0_0000_00a4 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:2				Reserved
1	EnableRdECCCapture	RW	0	When asserted the CSRs R_DdrxDdiRdECCCapture0-1 will store the value of the ECC field of the last read sent out on the CSW bus. This should only be enable during DDR DLL calibration, and not during normal operation where more than one read can be outstanding at a time.
0	ClearRdECC	RW	0	When asserted causes R_DdrxDdiRdECCCapture0-1 to clear

8.4.8.44 R_DdrxDdiRdECCCapture0

Register

R_DdrxDdiRdECCCapture0

Address

0x0_0000_00a8 (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24	Data3ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data3_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)
23:16	Data2ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data2_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)
15:8	Data1ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data1_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)
7:0	Data0ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data0_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)

8.4.8.45 R_DdrxDdiRdECCCapture1

Register

R_DdrxDdiRdECCCapture1

Address

0x0_0000_00ac (plus base address)

Bit	Mnemonic	Access	Reset	Definition
31:24	Data7ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data7_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)
23:16	Data6ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data6_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)
15:8	Data5ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data5_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)
7:0	Data4ECC	R	0	Stores the ECC value of the last read data driven out on ddr_coh_Data4_c2a[71:64] (Cleared by R_DdrxDdiECCCaptureEnable_ClearRdECC)

8.4.9 Register Allocation

This section instantiates two copies of the configuration registers for the two instances of DDR (DDR0 and DDR1)

8.4.9.1 Ddr0

Register

R_Ddr0* : R_Ddrx*

Address

0xE_4800_0000-0xE_48FF_FFFF

8.4.9.2 Ddr1

Register

R_Ddr1* : R_Ddrx*

Address

0xE_5800_0000-0xE_58FF_FFFF

8.4.10 Vregs_End_Of_Decl

8.4.11 DDR Performace Events

The following events are trackable by DDR statisticall event counting

Enum

DdrxEvent

Attribute

-descfunc

Constant	Mnemonic	Definition
8'h00	CYCLES	Dclk cycles. Always counts.
8'h01	CAS	Number of Read and Write commands issued to DDR2-SDRAM. For analysis studies on the use of auto-precharge tests can be run with R_DdrxDdiMifCfg1_AutoPch = 0. The difference (CAS - RAS) gives the total number DRAM accesses that hit on an open page within a bank. ((CAS - RAS) / CAS) gives the ratio of total page hits over total DRAM accesses.
8'h02	RAS	Number of Bank Activate commands issued to DDR2-SDRAM.
8'h03	MEMRD	Number of reads issued to the DIMM.
8'h04	MEMWR	Number of writes issued to the DIMM
8'h05	MULTRDBIDS	Cycles with more than one read request bidding for DDC.
8'h06	MULTWRBIDS	Cycles with more than one write request bidding for DDC.
8'h07	RDANDWRBIDS	Cycles with at least one read and one write request bidding for DDC.
8'h08	POWERDOWN	Number of cycles in powerdown.
8'h09	NEXM	Number of attempted accesses to non-existent memory. (These are software errors which could cause data corruption).
6'0a-8'hff		Reserved

8.5 DDC Section - DDR2 SDRAM Controller IP Block

The DDC section contains a version of NorthWest Logic's DDR2 memory controller customized for low latency. The read return data path has been removed. In our system, the core will pull read data directly out of the DDR2-PHY. The delay in the addres/CMD path has been reduced.

Specifications can be found in the project tree at:
.../hw/ip/northwestlogic/release_###/documentation/
DDR2_SDRAM_Controller_Core_Datasheet##.pdf
SiCortex_DDR2_Custom_Interface_Addendum##.pdf
- denotes version numbers, which may be different between the files and parent directory.

8.6 DDD Section - Datapath interface to PHY

DDD interfaces to the DDR2-PHY for extracting read data out of the PHY's read data fifo. DDD also replicates control signals from DDC into copies which are pitch matched to the individual PHY datapath slices.

Whenever DDP writes the first subcell of an entry of its read data fifo it toggles a signal which is sent to DDD. DDD synchronoizes this signal and begins pulling data out of the PHY's read data fifo and drives it out onto the

CSW bus (setting R_DdrxDddRdDelay_DelayFifoReadOut = 1, will an extra cclk cycle of latency before the data is read out of the fifo (this is not needed, but provided as a debug hook)). DDD runs on the CCLK, but can keep up with the rate that data is written into the read data fifo since it can pull out of the fifo to utilize CSW bandwidth through pipelining onto the 72 B CSW bus while the fifo input rate is at the DCLK but the width is only 16B wide (8B each on rising and falling of DCLK). When DDD causes the assertion of ddr_coh_DataValid_c2a it grabs the CSW bus (i.e. - DDR does not need to abitrate for CSW access).

8.7 DDP Unit - DDR2 SDRAM PHY IP Block

8.7.1 Overview

The DDP unit contains the DDR2-SDRAM PHY, which is a hard macro provided by designed by Esilicon. Some block diagrams and timing diagrams are located in the project tree under .../hw/ip/esilicon/doc/ddr2_phy_diagram_v#.pdf where # is the latest released version number.

8.7.2 Clocks

DDP receives two clocks DCLK and DM90CLK which is shifted minus 90 degrees relative to DCLK (i.e. DM90CLK is 1/4 cycle earlier). Both of these clocks originate from the one of the main PLL instances. Note that the PLL provides a pll_clock and pll_clock90 output which is shifted by positive 90 degrees, so DCLK will be driven by pll_clock90 for all of DDR/DDP and DM90CLK will be driven by pll_clock.

The clock which DDP drive to the DIMM is based on DM90CLK. The phase shift between the two clocks is used by the PHY in the write path logic to DDR2 spec requirement of DQS being shifted relative to DQ during writes.

8.7.3 Address and Command Path

DDP flops all address and command path inputs synchronously on the DCLK. These signals are then driven out the output pad to the DIMM. The command and address signals are:
A[15:0] - Address
BA[2:0] - Bank address
RAS_L - RAS command line
CAS_L - CAS command line
WE_L - WE command line (write enable)
CS_L[3:0] - CS command line (chip select (really rank select in our case))
ODT[3:0] - On-Die Termination
CKE[1:0] - Clock enable

8.7.4 Write Path

DDP flops all of its write path inputs synchronously on the DCLK. Some the of the write path signals are then flopped with DM90CLK. Please see ddr2_phy_diagram_v#.pdf for logic diagrams. During writes DQS is driven out 90 degress later than DQ.

8.7.5 Read Path

DDP's read return path is customized to reduce read return latency. Read data returning from the DDR2 DIMMs (DQ[71:0]) have an associated strobe clock DQS[8:0]. There are a number issues which need to be handled before the DQS can be used to capture the associated data. Firstly, because DQS is a bidirectional bus (driven by us during writes and driven by the DIMM for reads) it needs to be filtered so that it is doesn't cause false data capture due to it toggling during writes or toggle due to noise when it is undriven. Secondly, DQS needs to be shifted so that it lines up with the data eye so that data can be correctly captured. The read datapath is repeated 9 times corresponding to the 9 bytes of read data per read data received in parallel.

In order to filter DQS, the DDR2-PHY needs to identify preamble and postamble of the read data transfer. The start of the read preamble is defined as 1 clock prior to the first rising edge of DQS furing a read burst, with no external delays (DQS aligned to CLK_M90). The read postamble ends 1/2 clock after the last falling edge of DQS during a read burst, assuming no external delays (DQS aligned to CLK_M90). The NWL controller (DDC)

sends signals to DDP to identify the timing of the preamble (see logic diagram and associated timing waveforms for CTL_DQSA_PREAMB_ENABN and CTL_DQSB_PREAMB_ENABN on ddr2_phy_diagram_v#.pdf (location of this file is provided above in the overview subsection), also see the description of phy_dqs_preamble_en_n_a and phy_dqs_preamble_en_b in NWL's DDR2 SDRAM Controller Core SiCortex Custom Interface Addendum). These signals are combined to create DDO_DQS_PREAMB_ENABN, which is then sent through a dummy instance of the differential I/O cell used for DQS to match delay variation due to PVT changes seen by DQS. The preamble enable then goes through a slave DLL which compensates for the board trace length round trip delay between the ASIC and DIMM (the delay setting for this DLL is controlled per byte lane by the CSRs R_DdrxDdpDLLLane#_Slave0Adj). The output of the DLL enables the PHY to receive the DQS strobe and starts a 4 cycle counter which keeps the enabling the PHY to recieve DQS (the counter works becuase all reads are full 72 B (4 cycle) reads).

Ideally, after DQS is filtered, its timing will match that of the DQ input after it has gone though similarly matched logic. It is then necessary to delay the DQS by approximately a quarter cycle so that it can be used as a capture clock for DQ. This delay is obtained from a second slave DLL (the delay setting for this DLL is controlled per byte lane by the CSRs R_DdrxDdpDLLLane#_Slave1Adj).

The captured read data is place into a fifo which lives in the DDR2-PHY. The fifo is 4 entries deep, where each entry is 72 B wide. Each fifo entry has 8 sub-cells corresponding to each of the 8 data sub-transfers associated with a full 72-B read. Whenever DDP writes the first sub-cell of a fifo entry it tells sends a signal to DdrDdd to signify that it is safe to start pull data out of the next fifo entry. After proper sychronization, DdrDdd starts pulling data out of the PHY.

8.7.6 DLLs

Each of the 9 bytes lane of the a PHY instance includes an embedded analog DLL module from True Circuits based on their Part: TCI-TN90G-DDRLDLL. Each module contains one master DLL and two slave DLLs. Detailed information is located in the project tree at .../hw/ip/esilicon/release_11_19_05/dll_090g, in particular the document TCITSMC009DDRDDLLA1_guide.txt is very informative.

Each DLL module contains 1 master DLL and two slave DLLs. The master DLLs

Reference input frequency range: 93MHz - 465MHz

Slave delay adjustment range: 0% - 100% of reference clock

Number of slave adjustment steps (MADJ) - 160 (See below, DLL Master Adjustment section as Sam Stewart at Esilicon provided different info)

Slave delay equation - Tf + [(ADJ + ADJ_offset)/MADJ] * Tref

Fixed delay offset (Tf) (nom) - 90ps (this delay is cancelled by the match cell used for the DQS shift path.)

Fixed code offset (ADJ_off) - 34 steps

8.7.6.1 DLL Master Adjustment

According to information provided by Sam Stewart at Esilicon, the MADJ setting is frequency dependent. Verilog simulations of the PHY seem to corroborate this.

MADJ_MAX = (160 * 465Mhz) * Tref

This implies the following settings should be used for R_DdrxDdpDLLLane#_MasterAdj:

DCLK = 400 MHz (2.5ns) => MADJ = 184

DCLK = 333 MHz (3ns) => MADJ = 224

DCLK = 267 MHz (3.75ns) => MADJ = 252 (The formula says 279, but the MADJ is 8 bits wide (caps at 255))

8.7.6.2 DLL range calculations for Slave0 (DQS preamble enable DLL to match board trace length to memory)

DLL slave 0 adjustment range: 1-134.

8.7.6.3 DLL range calculations for Slave1 (DQS 1/4 cycle delay DLL)

DLL slave 1 adjustment recommended settings:

DCLK = 400 MHz (2.5ns) => ADJ1 = 12

DCLK = 333 MHz (3ns) => ADJ1 = 22

DCLK = 267 MHz (3.75ns) => ADJ1 = 29

Slave 1 setting = ((MADJ) / 4) - ADJ_offset, where ADJ_offset is the ADJ fixed code offset of 34 steps. Tf has been compensated for in the design.

8.7.7 I/O pads

The I/O cells used DDR2 are from ARM's 90nm 1.2Gbps DDR1/DDR2 Combo Library for TSMC G. These ahve 1.8V drive, 1.0V Core interface for DDR2.

8.7.7.1 Impedence Calibration

The I/O cells include pull-up and pull-down impedence for driver strength setting and On-Die-Termination.

Chapter 9

Counters, Performance Counters, & OCLA Overview

[Last modified: $Id: counters.lyx 31059 2007-01-30 21:16:09Z pholmes $]

9.1 What's Available

The Ice9 chip provides various ways to gain information on internal events and status. The SCB Bus provides access to internal status and counters to MSP (and SSP) from outside an Ice9, as well as to the 6 processors within Ice9. Processor code can read CPU Counters. And internal signals can be driven to an Ice9 external pin.

This status information is provided by SCB Registers, the SCB Performance Counters mechanism, and OCLA (On Chip Logic Analyzer). Performance Counters and OCLA can be used in various ways.

Simpler methods of gaining visibility take less configuration effort than the more complicated methods. In order by increasing complexity, these methods of gaining visibility into Ice9 are:

- SCB register "good" and "bad" status bits within various sub-blocks of Ice9, many of which can cause interrupts.
- SCB register counters within various sub-blocks of Ice9.
- CPU Performance Counters, 2 in each MIPS core.
- SCB Performance Counters used to get up to 2 configurable 32-bit counters.
- SCB Performance Counters used to get up to 256 statistical-percentage counters.
- OCLA driving internal signals out an Ice9 external pin.
- OCLA used to get a highly-configurable 12-bit counter.
- OCLA used to record a timeline of the times when an event occured.
- OCLA used to capture trace and values informations like a logic analyzer.

And of course you can use more than one of these at the same time. You could have the SCB register counters counting, at the same time that SCB Performance Counters is doing something, at the same time that OCLA is doing something. Let's look at each of these in more detail...

9.2 Status Bits

Various Ice9 sub-blocks have "good" and "bad" status bits that can be read from SCB registers.

For an overview on error conditions, and info on ECC errors, see the "Reliability, Availability, Serviceability, and Error Handling" chapter of the system hardware spec. This chapter can also be found under rev-control in **<project>/specs/system/Reliability/Reliability.lyx**.

The PCI-Express unit has a "Link Up" bit, the 6 Fabric Link units have "MissionMode" bits.

Most sub-blocks have "bad" status bits of various kinds. Most of these can be enabled to drive interrupts to the processors. Even when a particular interrupt is not enabled, the status bit for that condition is usually readable over the SCB Bus.

9.3 Counters

Some Ice9 sub-blocks have counters locally-implemented (within the sub-block) that can be read from SCB, counting normal and error type events. Some sub-blocks rely entirely on the SCB Performance Counters for any counting you may wish to do, and some have both their own counters as well as SCB Performance Counters hookup.

Locally-implemented counters are simpler to get information from than OCLA or SCB Performance Counters, requiring no configuration ahead of time, except in some cases they should be cleared at the appropriate step in boot process. Furthermore, they're always "on", giving a true count of their particular event.

In the DMA sub-unit, a philosophy was taken that if counting was needed, DMA microcode could do the counting and store the values in memory.

Fabric Switch counters are 32-bit, but counters in the Fabric Link are much smaller.

These counters may not have been verified as well as the main functionality of the chip, depending on the sub-block. For some counters the count may not be exactly what would be literally correct during complex error conditions. But in general, during error-free conditions the error counters will remain zero and the good event counters will count correctly. And in general, during simple error conditions the error counters will count their respective errors correctly.

As of September 2006, Link-unit counters have been verified for small counts, but not for large counts or rollovers. Nuances about their counts are documented in the Link Spec **<project>/specs/ice9/link/link.lyx**.

As of September 2006, Fabric Switch counters have been tested as correct during good traffic and simple errors, although during complex errors or periods of time not processing traffic the counts may be off. Fabric Switch counters are documented in **<project>/specs/ice9/fabric/fabric.lyx**.

9.4 CPU Performance Counters

Each of our 6 embedded MIPS cores has 2 configurable Performance Counters within it.

See the MIPS Spec **<project>/hw/cpu/opal_2_3/docs/MD00012-2B-5K-SUM-02.08.pdf** section 6.22 "Performance Counter Register". Read this for the mechanism of how to use these counters, but read the "Processor Segments" chapter of our Chip Spec for the list of events.

In the "Processor Segments" chapter **<project>/specs/ice9/processor/processor.lyx** see section "CPU Performance Counter Events". Note differences between ICE9A and ICE9B.

9.5 SCB Performance Counters

836 different events or conditions are wired to the SCB Performance Counters mechanism, coming from many sub-blocks in Ice9, with strong emphasis on the processors themselves. There's a good list of within-processor events to count, separately selectable for CPU0 through CPU5. In addition to these events wired directly to the SCB system, much of the OCLA triggering system is also available as events for SCB Performance Counters.

SCB Performance Counters require configuration in order to be used, but it's much simpler to use than OCLA.

SCB Performance Counters are 32-bits, many more bits than the counters in OCLA.

Not only can you choose from that long list of events, but you can condition any event by another event, counting only "if AND" the other event, or "if AND NOT" the other event.

You can choose between "how many clocks was it high for" and "how many times did it go high for awhile", or even "how many times was it high for more than N clocks". Collecting that last version for more than one value of N, you can gather histogram information.

Tests causing each event (that's wired to the SCB Performance Counters) have *NOT* been written as of October 2006, so some events may not work correctly. More to the point, although I expect most events to work correctly and count what you'd think they count, in a few cases the name of the event may not mean what you first think it does. When in doubt, ask the sub-unit designer what's being counted (or asserted) by that event signal.

There are cross-connections in both directions between SCB Performance Counters and OCLA, but those connections are not required for use of either. To keep SCB Performance Counters configuration simple, first see if the events you need are directly available in the AllEvents list. If not, then look at what events OCLA could

provide. Accessing OCLA events for counting is much simpler than the full use of OCLA, no OCLA LAC program is needed.

See the "Serial Configuration Bus" chapter of the chip or hardware-system spec. This chapter can also be found under rev-control in **<project>/specs/ice9/chipSCB/chipSCB.lyx**. There's a lot in that chapter, so look for the "Performance Counting" sub-section, and then the later "Performance Counting Registers" sub-section.

In our Chip Spec there is no one list of all the events which can be counted by SCB Performance Counters. The best place to look for a nearly-complete list is in the software defines extracted file. As of January 2007 software defines for these are **<project>/sw/include/sicortex/ice9/ice9_all_spec_sw.h** as enum Ice9_EnumAllEvent.

The majority of SCB Performance Counters events are from inside the 6 processors. The list of "from the processors" SCB Performance Counters events is found in the "Processor Segments" chapter **<project>/specs/ice9/processor/proc** sections "SCB Performance Events" and "SCB Performance Core Events". Note Ice9A vs Ice9B differences. This list is duplicated 6 times, once for each MIPS processor.

OCLA events are not listed in the extraction. Although the hardware exists to count OCLA trigger-block events in SCB Performance Counters, it is not actively-supported or documented at this time.

The SCB Performance Counters mechanism can be used in 2 quite-different ways, for "ordinary counters", or for "statistical percentage counters", as described below.

9.5.1 Ordinary Counting with SCB Performance Counters

If you want "a count of how many times something happened" for one or two of the many events wired to the SCB Performance Counters, you can configure this mechanism to dwell on those events continuously, giving you a "full count" of how many times those events occurred.

There's a limit of 2 events at a time.

If you want an event conditioned by another, then those 2 events have already used-up your limit of watching only 2 events at a time.

You may wish to be careful to remain off-of the SCB Bus during the time-period you're interested in. Any SCB writes or reads create short "black-out times" when your events may occur but not be counted.

Prior to the time-period of interest, use SCB-writes to configure SCB Performance Counters for the events you want, and for a large dwell-time, and no incrementing of the bucket-number. Then, after the time-period of interest, use SCB-reads to get your counts.

Events coming from clock-domains other than cclk (like FSW) will be counted correctly.

9.5.2 Statistical Counting with SCB Performance Counters

Your choice of up to 256 of the available events (AllEvent and OCLA events) can be scanned with a given configuration of SCB Performance Counters.

In this style-of-use the goal is to get an estimate of "activity density" or "statistical percentage-of-time" of the events. For each selected event-signal you will be able to get a rough estimate of what percentage of time that signal was true.

With this information you could compare different events to see which was occuring more often or more of the time. When tuning or diagnosing performance, you can see percent utilization of an Ice9 sub-block, or an interface from one sub-block to another.

The SCB Performance Counters mechanism scans across the configured events, dwelling the same amount of time on each. After a period of scanning, you read the counts for each event. You can compare them, or divide these counts by the number of cclks spent watching for each, to get a percentage-of-time asserted.

This style of use does not get you a "full count" of events, because the mechanism was scanning across events. For any one event, most of the time that event wasn't being watched.

This style of use ***is*** protected against black-out times when SCB writes or reads are taking place. The dwell time of watching for an event doesn't count time periods when SCB writes or reads are happening.

To get good statistics, the activity of interest should be more-or-less in a "steady state", and then SCB Performance Counters should be configured to dwell long enough on each event to get a representative sample, as described in the "Serial Configuration Bus" chapter.

9.6 OCLA

OCLA (On Chip Logic Analyzer) was designed to capture values of many signals in response to a simple or complex trigger event, but it can also be used in simpler ways. OCLA is provided with a large number of signals and busses from many Ice9 sub-blocks. With these you can form simple or complex triggers, and select which groups of signals you wish to capture in Collector Blocks for later viewing.

OCLA can also trigger on up-to 2 of the many events provided to SCB Performance Counters, and can combine those events with OCLA's own events in an AND-OR-delay manner to form triggers. But it's simpler and usually adequate to use OCLA's own large selection of trigger signals. If you configure OCLA to use SCB Performance Counters events, this "ties up" the SCB Performance Counters mechanism, in that any counting done by SCB Performance Counters must be on those same events. Furthermore, you must manage your SCB writes and reads to avoid missing events you wished to trigger on. No such management of SCB accesses is needed if you use OCLA's own trigger signals.

The OCLA Spec is the "On Chip Logic Analyzer" chapter of the chip or hardware-system spec. This chapter can also be found under rev-control in **<project>/specs/ice9/chipocla/chipocla.lyx**

OCLA is fairly difficult to program. Expect a learning-curve. Your first OLCA program will likely not work at all. Example programs have been written and made to work in simulation for each of the Ice9 sub-blocks containing OCLA, for many trigger signals, and for various styles-of-use of OCLA. When writing a new OCLA program it's recommended that you get it going in simulation first, then transfer it to the lab. Even experienced OCLA-programmers often resort to simulation-waves to debug a non-working OCLA program. The lab, of course, doesn't have such visibility.

The OCLA wiki page **http://apollo.sicortex.com/swiki/OclaVerification** lists working example programs and where to find the code for them. You can use the Makefile there to create a Diagnostics "dash" perl script with the configuration of any of these programs. This perl script gives you the same OCLA configuration in the lab as was in the simulation test. These perl scripts are fairly readable and can be edited if you know OCLA well enough.

9.6.1 OCLA Driving an External Pin

OCLA can be configured to drive any 1 of 100's of internal signals to Ice9 external pin "sys_ocla_trig".

The signals to choose from are those leading into the OCLA Trigger Blocks, as described in the OCLA Spec.

The occurance of SCB Performance Counters events may also be driven out this pin.

Logical combinations if signals and pattern-matching on busses can be combined to determine when to drive this pin.

This can be useful to: (a) gain visibility inside the ASIC as to whether or how-often an internal event is happening, (b) trigger lab logic-analyzer equipment at the correct time to capture external busses data.

To do this a small OCLA LAC program is required, as well as configuring one or more Trigger Blocks.

There will be a fixed multi-clock delay, of some 20 to 40 nSec depending on the signal, between activity on your selected signal, and that same activity on "sys_ocla_trig".

Signals from the FSW unit, and events from SCB Performance Counters, will be distorted due to clock-domain crossings, and the need to stretch short pulses so they don't dissapear as they enter the cclk domain and pass through OCLA. Isolated high pulses will not be lost, but sometimes 2 closely-spaced pulses from FSW or SCB Performance Counters will merge into one pulse.

The fastest oscillation of this output is at 1/2 cclk frequency. Quality of viewed waveform will depend on how well the signal is kept close to a ground signal as it passes from ASIC, through board, into scope probe, into scope. With a couple inches of distance along the way not twisted with ground you can still tell the difference between actual pulses driven high and ringing/reflections.

9.6.2 OCLA as a Counter

One simple use of OLCA is as a counter.

Only a very simple LAC program is needed, but even so, it's usually less configuration to feed the signals or triggers to SCB Performance Counters, and do the counting there. OCLA is more flexible, but SCB Performance Counters is pretty powerful. If you wish to count one trigger qualified by another, SCB Performance Counters can do that. If you wish to count one trigger qualified by a delayed or advanced version of another trigger, SCB Performance Counters can do that, with the delays being applied in OCLA LAC before the triggers are sent to SCB Performance Counters.

SCB Performance Counters are 32 bits whereas OCLA counters are only 12 bits. Fortunately OCLA counters have a sticky overflow-bits indicating when over 4095 counts occured.

You can have a 24-bit counter by nesting OCLA's two counters in OCLA LAC program loops, but you get a slightly imprecise count, because it's not watching for the event every time you "carry" from the lower-bits counter to the upper-bits counter.

One motivation to count in OCLA rather than SCB Performance Counters is that SCB Performance Counters has black-out periods (missing counts) whenever an SCB write or read is in progress.

Another motivation to create a counter in OCLA is if SCB Performance Counters is already in use, or if you wanted more than 2 continuously-counting counters. 2 continuous full-count counters in SCB Performance Counters plus one in OCLA gives you 3 at once.

2 in SCB Performance Counters plus 2 in OCLA gives you 4 at once, but OCLA cannot increment both of OCLA's counters in the same clock, so you'd have to decide which count gets incremented and which doesn't if both events happen at the same time. If the 2 events are known to not happen on the same clock, there's no problem. If the events are sparse and unrelated you could just accept one of the counts being inaccurate. If the events would predictably occur on the same clock, you could delay one of them with LAC delay regs.

The real power of counting with OCLA trigger blocks is configurability. You can "design your own counter"! OCLA can be configured to count an AND-OR combination of many signals, even delayed signals! OCLA can also be configured to only count when an address, state-encoding, or packet header information on a bus matches one or more values or address-ranges. Much of this can be fed through to SCB Performance Counters, with the counting done there, but the full AND-OR combination flexibility is only available by counting in OCLA.

9.6.3 OCLA as a Times-of-Occurance Recorder

Using OCLA's free-running-counter to collect time-stamps in a collector block, you can get an "event timeline" of any OLCA-trigger event. As stated before, this event can be an AND-OR combination of signals or delayed signals, including pattern-matching on addresses, state-encodings, or packet headers.

Up to 1024 event-timestamps can be collected. You lose any events after that.

The free-running counter is 32-bits, so timestamps for up-to 2**32 cclks (16 seconds) are non-ambiguous. An unambiguous time-record for more than 16 seconds (with no upper limit in time), for 1024 or less occurances of the event, can be had by writing some watching software for one of the processors that periodically reads some OCLA registers.

You usually get no useful logic-analyzer type collection of values occurs when using OCLA in this manner, all you get is a series of timestamps.

9.6.4 OCLA as a Logic Analyzer

The full use of OCLA's capabilities is to collect values of many signals and busses in response to a simple or complex trigger event. The OCLA Spec is where this is described in detail, but here are a few highlights:

Up to 1024 cclks of activity may be collected.

This collection can be done for one period of time of 1024 cclks, or multiple smaller time periods can be collected.

A "qualification" feature allows some data (but not all) to be collected "only when valid", which is very efficient. This allows many short events, or single-clock events over a long stretch of time to fit in a 1024 entry collector block.

A period of collection can programmed to be mostly prior to the trigger, centered around the trigger, or after the trigger.

Activity can be collected in more than one sub-block of Ice9 at the same time.

Although there are many choices of sets of data to collect, they represent only a small fraction of the signals and busses in the sub-blocks of Ice9. We did the best we could to choose "likely to be useful" data to wire-up to OCLA, but by hindsight we already see we could have made some better choices. Hopefully what you need will be there.

Chapter 10

Serial Configuration Bus

[$Id: chipSCB.lyx 50693 2008-02-07 16:01:46Z wsnyder $]

10.1 Overview

The Serial Configuration Bus (SCB) is a small serial bus used to interconnect software programmable registers (aka CSRs or slow I/O registers) throughout the chip.

The SCB is controlled by the SCB Master block. The SCB Master (SCBM) interprets CPU reads and writes, and converts them to a serial bus. The serial bus is driven to the first in a ring of SCB Slaves (SCBS). It eventually reaches the desired slave, which performs the read or write and drives the data further along the ring and finally back to the SCB Master.

The SCB also implements performance counters, which statistically sample monitoring points across the design.

10.2 Specifications

- Up to 128 slaves.
- 32-bit data.
- Up to 24 bits of configuration register address space per slave.
- Low-cost 3-signal interconnect.
- SysChain interface for module processor access to all slave registers.
- Synchronous clocking in each required clock domain.
- Standardized Slave interface, for easy instantiation.
- Low cost reset of all I/O register state.
- Performance sampling interface, with up to 256 different events per slave.
- All of OCLA events, plus enum AllEvent available for performance counting.
- Any of the performance events visible at OCLA, for logic analyzer triggering.
- System Manager interface registers for LEDs, Attention, chip number, etc.

10.3 Differences, Bugs, and Enhancements

10.3.1 Product and Chip Pass Differences

1. ICE9B returns a different product (ICE9B) and/or revision (ICE9A1 vs ICE9A0) when reading R_ScbChipRev.
2. ICE9B has reduced latency accessing the SCB's own registers.

3. ICE9B adds a interrupt/attention for when the Chip<->Msp channel is ready for transmit.

4. ICE9B adds R_ScbDInt to replace the SysChain R_SysTapDint register, see bug2223.

5. TWC9A returns a different product (TWC9A) and/or revision when reading R_ScbChipRev.

6. NEED IMPL: TWC9A supports 64 bit SCB slaves and 64 bit registers, see bug4619.

7. TWC9A adds R_ScbDInt_SendDInt6, R_ScbDInt_Cpu6DM, R_ScbAtnInt_Cpu6DMMask, and R_ScbAtnInt_Cpu6DM to support CPUs 6-9.

8. TWC9A fixes reads to fast DDR clock registers returning the wrong results after a CCLK register read, bug4331. Earlier chips required a dummy read between such read sequences.

9. TWC9A will skip sampling bucket pairs where R_ScbPerfBuckets_Event == AllEvent_INVALID. This is backward compatible with other products, which should use that encoding for invalid buckets. bug4265.

10.3.2 Known Bugs and Possible Enhancements

1. In ICE9A and ICE9B, all SCB accesses must be done with 32-bit accesses. Using a 64-bit read/write to access them will put return/write data in the wrong half of the quadword, not simply return or write half of the data.

2. Decouple the SCB CPU#_P[01] events from the CPU performance counter domain (U/S/K), perhaps with new domain bits.

3. SCB performance counts from Ocla TrbC blocks depend on the TrbC configuration, this could be simplified. bug1717.

4. R_ScbPerfEna should have a way to stop immediately, without corrupting, for interrupt handlers. Perhaps add a _Pause bit that stops on current bucket and partial interval. We'll also need to make the partial interval programmable so context switches can reprogram it.

5. R_ScbPerf* registers should be writable without needing to stop sampling.

6. R_ScbInt should indicate what bucket(s) have caused the overflow, to save software from having to read the entire count ram on each overflow, bug2164.

7. R_SysTapMsp transactions should be double buffered, as the Msp decision loop is quite slow.

8. R_ScbInt like most of the other blocks in the chip contains the interrupt state before masking. This requires the interrupt handler to read (or cache) R_ScbIntMask before dispatching interrupts.

10.4 Block Diagram

10.5 SCB Master Ports

Signal Name	In/Out	From/To	Description
pmi_scb_req_cr	In	Pmi	Pmi Scb request pulse. Pulsed to request a Scb transaction, _wr, _addr, and _wdata are valid until acknowledged.
pmi_scb_addr_cr	In	Pmi	Pmi Scb request read/write address.
pmi_scb_wr_cr	In	Pmi	Pmi Scb request write, not read.
pmi_scb_wdata_cr	In	Pmi	Pmi Scb request write data.
scb_pmi_ack_cr	Out	Pmi	Pmi Scb return acknowledge. Pulsed to indicate completion of transaction, and _rdata is valid.
scb_pmi_rdata_cr	Out	Pmi	Pmi Scb return read data.
scb_csw_ScbInt_ca	Out	Pmi	Pmi Scb interrupt. Asserted while interrupt requested.
scb_chaino_dat_*r	Out	first SCB Slave	Serial SCB chain output (one per clock domain)
chaini_scb_dat_*r	In	last SCB Slave	Serial SCB chain input (one per clock domain)

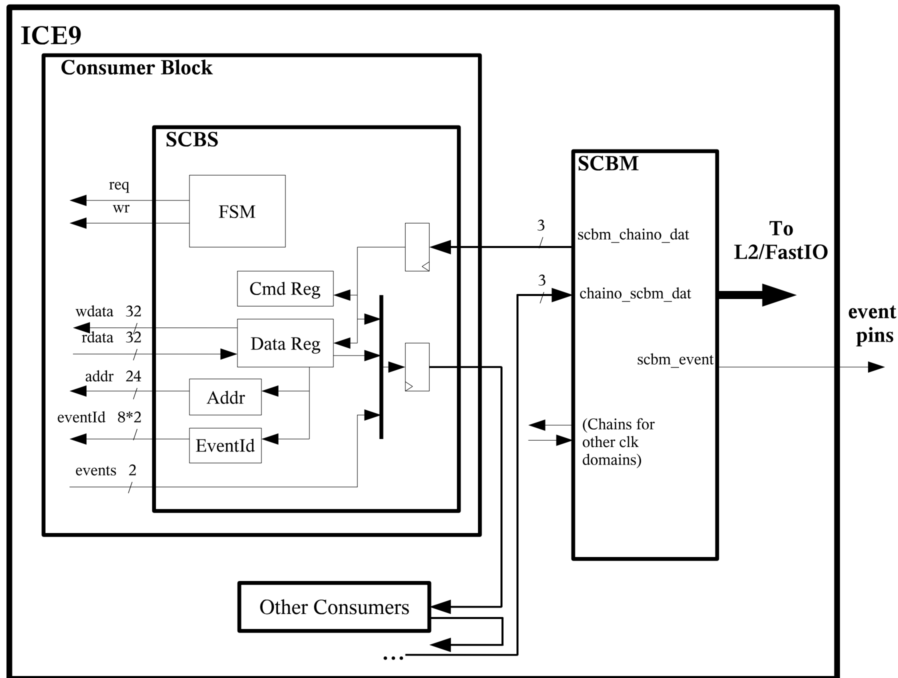


Figure 10.1: Scb Overview

10.6 SCB Slave Ports

The SCB Slave is a standard Verilog/SystemC module that is instantiated by blocks to decode the serial bus into connections for the local block's register logic.

Signal Name	In/Out	From/To	Description
chaini_scbs_dat_r[2:0]	In	previous SCB Slave	Serial SCB chain input.
scbs_chaino_dat_r[2:0]	Out	next SCB Slave	Serial SCB chain output.
scbs_x_active_r	Out	slave user	Transaction active. May be used as a clock gate for slave logic that only needs to be active during SCB activity. Asserted starting with the initial req_r assertion through acknowledgement, and during event counting.
scbs_x_req_r	Out	slave user	Read/write request pulse. Indicates address and write data is stable.
scbs_x_addr_r[23:0]	Out	slave user	Decoded address, for register accesses and selection of sample point.
scbs_x_wr_r	Out	slave user	Write/ not read. Asserted for writes, deasserted for reads.
scbs_x_wdata_r[31:0]	Out	slave user	Write Data. 32-bit data bus for writing.
x_scbs_ack_r	In	slave user	Read/write acknowledge pulse. Pulsed to acknowledge write, or read data is ready.
x_scbs_rdata_r[31:0]	In	slave user	Read Data.
x_scbs_id[6:0]	In	slave user	Identity. Specifies constant 7 upper address bits that must match address to accept SCB transaction. See 16.6.6.
scbs_x_counting_r[1:0]	Out	user events	Asserted when the events are being counted by the SCBM. May be used to gate latching of last-event addresses, etc.
scbs_x_eventId0_r[7:0]	Out	user events	Event number to route to x_scbs_event[0].
scbs_x_eventId1_r[7:0]	Out	user events	Event number to route to x_scbs_event[1].
x_scbs_event[0]	In	user events	Count bit A. Level asserted to count event on eventa_r for this cycle.
x_scbs_event[1]	In	user events	Count bit B. Level asserted to count event on eventb_r for this cycle.

10.7 Custom/Large Structures

Name	Size	Description
ScbCntRam	256x50 1rw	Counting RAM, size based on number of sampling points, so easily negotiable.

10.8 I/O Operations

The SCB master connects to the system via the PCI Host interface, which receives I/O read and write transactions from the CPUs. When the SCB master detects an I/O write to its 32-bit address space, it initiates a SCB I/O write operation on all of the SCB busses.

The address and data are shifted onto the SCB buses. One of the 128 SCB slaves decodes the address, and asserts a request to the SCB slave user's logic. The user logic writes the register and asserts a strobe back to the SCB slave. The slave returns the acknowledge back over the SCB bus to the SCB master.

On a read, the address is shifted onto the SCB bus, and is decoded by one of the SCB slaves, which asserts a read request to the SCB user logic. The user logic reads the registers and returns the read data and acknowledgement to the SCB slave. The SCB slave shifts out the acknowledgment and data back to the SCB master, who returns it to the system bus.

10.8.1 No responder

When a SCB slave sees a transaction to its address space, it asserts Cmd[0] back to the SCB master. Should no slaves respond in this way across all of the domains, the SCB master will acknowledge the transaction itself (since no slave will ever respond.) On writes, this means the write will be silently dropped. On reads, the return data will be zero.

10.8.2 Approximate Latency

The approximate read latency of SCB operations is calculated below. Currently the sclk is the both the slowest chain and the chain with the most loads (8). This yields a minimal latency estimate of 210 ns.

Who	How Much	Description
Cpu	2 pclk	Read issue latency
Fsw	~2-5 cclks	Latancy across Fsw And Cac
Pmi	~2 cclks	Pmi Latency
Scbm	~8 cclks	Scbm Overhead
Scb	20 Xclks + #slaves	Time to clock command. This is the maximum across all clock domains.
Device	3++ Yclks	Time for slave to respond to request for data.
Fsw	~2-5 cclks	Latency across Fsw and Cac. (Due to bus-stop organization, this is likely to be smaller if the above Fsw latency is large, and vice-versa.
Cpu	1 pclk	Read return latancy

10.8.3 Software Notes

The SCB registers must be accessed with 32-bit load/store operations. Other size operations are not supported.

10.9 SysChain Interface

The registers on the SCB bus may be accessed over the SysChain interface. This may be done at any time; it is round-robin arbitrated with the normal Pmi path.

10.9.1 SysChain Access Requirements

To access SCB registers via the SysChain bus:

1. SCBM/BBS reset must be deasserted. SCB slaves may still be in reset.

2. All clocks with SCB chains must be running, not just the cclk and destination slave clock.

3. Software must ensure that one SysChain write/read completes with "done" before the next is launched, or must request a reset between transactions.

4. An old transaction may be shifted out simultaneously with a new command shifting in.

10.9.2 SysChain SCB Write

To write a register on the SCB chain, the address and data is prepared in the R_SysTapScb structure. The write and go bits are set, and the structure is shifted into the SCB SysChain interface. The SCB will decode the command and see the go bit set. It then performs the IO write as described above. On completion, the command register may be shifted out; the go bit will now be clear, and the done bit will indicate if the write was completed.

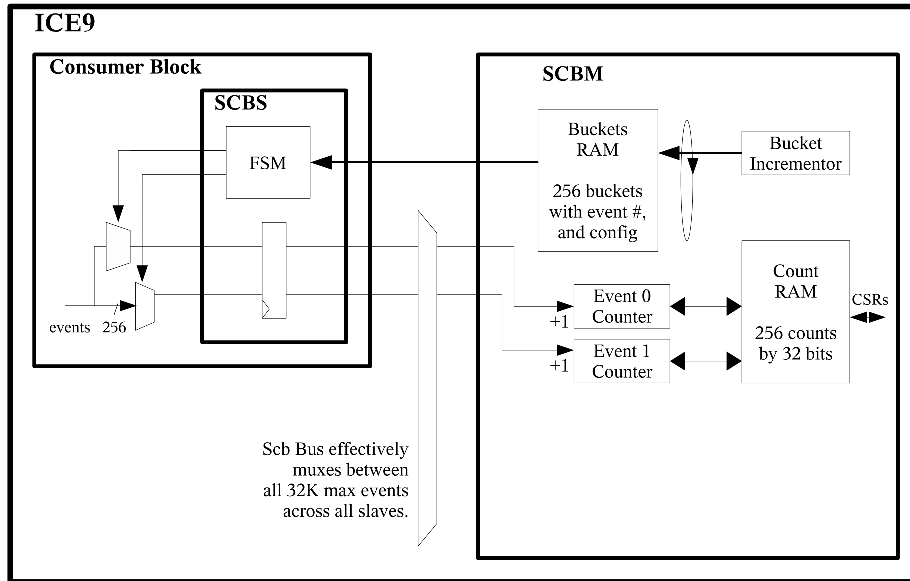


Figure 10.2: Scb Performance Counting

10.9.3 SysChain SCB Read

To read a register on the SCB chain, the address is prepared in the R_SysTapScb structure. The write bit is cleared, the go bit is set, and the structure is shifted into the SCB SysChain interface. The SCB will decode the command and see the go bit set. It then performs the IO read as described above. On completion, the command register may be shifted out; the go bit will now be clear, and the done bit will indicate if the read was completed; if so the data field contains the read data.

10.10 Performance Counting

When not being used for an I/O operation, the SCB bus may be used for counting events and performance monitoring.

10.10.1 True Counting

SCB Performance Counting can provide you a full count of how many times up to two events happened. You configure buckets 0 and 1 only, and don't enable incrementing to the next pair of buckets. Even if the SCB slaves selected are in a different clock domain from the SCB master, an accurate count of events at the SCB slave will be tallied. The only events you miss are those that occur during an SCB bus I/O operation, so you should be careful to manage SCB bus use during accurate counting.

10.10.2 Statistical Counting

Up to 256 events can be counted in a statistical manner, watching for each for an equal amount of time.

When enabled by R_ScbPerfCtl_Run, the SCB starts with bucket 0. The R_ScbPerfBuckets[0] register is loaded, which directs the SCB to select a given event number to sample into that bucket, see 10.17.9. In Twc9a+, if the event number is INVALID, the SCB skips the rest of this description and moves onto the next bucket.

The event number is shifted to all of the SCB slaves. The slave corresponding to that event then routes that event's state to the data wires, which propagates back to the SCB master. The SCB master increments a counter each cycle the data wire is true, thus calculating the number of cycles the event was true.

To allow for better debugging and tracking of cross products, the SCB can determine how long a signal was asserted on two such events at once, one on each of the two serial data wires. While R_ScbPerfBucket[n] is being counted, the event in R_ScbPerfBucket[n+1] is simultaniously being counted.

After a programmed delay in R_ScbPerfCtl_Interval, the SCB adds the event counter to the total in the R_ScbPerfBucket_Count[0] (and [1]) register, see 10.17.10. It then increments the bucket number by two and begins the process again with the event in R_ScbPerfBucket_Count[2] (and [3]).

In this way, over time, the SCB has a statistical average of how often each event occurs. To reduce sampling errors on events which are asserted for long times, 1K cycles seems a reasonable minimum sample interval per bucket. At this interval we can go through all buckets at 250Mhz * 2 events at once / 256 buckets / 1K cyc/event = 488 samples per second. (This ignores the minor overhead in switching between events, so the real figure is ~4% smaller.)

Once you have a count of events at an SCB slave in a different clock domain from the SCB master, if you want to calculate the percentage of slave clocks when the event was true, you must factor-in the ratio of clock speeds between SCB master and slave.

10.10.3 Counts Causing Interrupts

The software can configure interrupts when the event counters set a certain count bit number. For example, if R_ScbPerfCtl_IntBit==31, a interrupt will be raised exactly when an event causes its counter to count above 2^31. (Not while it is above 2^31, but when the event itself occurs.) Software then clears the interrupt.

Note the interrupt for event x overflowing may be signaled before R_ScbPerfCounts[x] is written with the overflowing value. Software should poll R_ScbPerfBuckNum in the interrupt handler to see it increment once if it relies on R_ScbPerfCounts[x] to indicate what bucket(s) overflowed.

10.10.4 OCLA Triggering

From SCB Performance Counters to OCLA:

Both of the final count wires, as seen by the SCB master, are routed to the OCLA. These two signals add to the large collection of things OCLA already has to trigger on. These provide OCLA the ability to trigger on any of the events SCB Performance Counters can count. But, in order to do so, SCB Performance Counters must be configured to dwell continuously on the one or two events that OCLA wants to see.

10.10.5 Events from OCLA

From OCLA to SCB Performance Counters:

All of OCLA's TRBC or TRBV triggers, and the raw signals from TRBVs, are available to SCB Performance Counters as events to be triggered on. These events are in addition to those listed in enum Ice9_AllEvent.

10.10.6 Arbitration

SCB I/O operations and event counting require the same SCB slave data wires.

To avoid conflict, when a SCB I/O operation occurs, the current event count will be suspended, the SCB I/O operation performed, and the same event count restarted from where the count ended. In the end, the event will have been sampled for the same number of cycles as if it had never been interrupted. The interruption may cause minor inaccuracies in the counting, but should be negligible given how infrequently SCB accesses will occur.

10.10.7 Software Notes

Each event is loaded into a 32 bit count register. To prevent overflow, these counters must be sampled at least every 4G/500 MHz = 4 seconds. (It is more typically 10,000 seconds, as in normal operation each event is only sampled for 1/2048th of the time, but the SCB may be programmed to count only a single event forever.) Software should sample significantly faster then this (once per second), and derive the rollover bits to present a 64-bit counter to the upper level application.

The best presentation to the user is probably as string-indexed values. The strings will be automatically extracted from the enum declarations in the specifications by the vregs package.

All performace registers are in a unique 64KB page to allow software to map only the performance counter physical page into user visible virtual address space.

10.10.8 Writing while Counting

Generally software should stop the counters before writing them. If, however, the counters are running, the table below describes the potential hazards. Note writing the same value never has an effect; the table only applies when the value to that field will change.

Register	Effect
R_ScbIntMask_*	Takes effect immediately. No hazards.
R_ScbIntReq_*	Takes effect immediately. No hazards.
R_ScbPerfCtl_NoInc	Takes effect at the end of the current interval.
R_ScbPerfCtl_IntBit	Takes effect at the beginning of the next interval.
R_ScbPerfCtl_Interval	When a count is in progress, changing the interval may make the counter overflow. Not recommended.
R_ScbPerfHist_HistGte	Takes effect immediately. If the bucket being sampled is using histogram, the count currently being calculated may spuriously count or lose a few events. Not recommended.
R_ScbPerfBuckNum_Bucket	If R_ScbPerfCtl_noInc is set, the written value will be used when the next interval begins. If R_ScbPerfCtl_noInc is clear, the written value, or 2 plus the written value may be used when the next interval begins.
R_ScbPerfEna_Ena	Writing a one has no effect, as counting is already running. Writing a zero requests disabling counting when the next complete round of sampling completes.
R_ScbPerfStat_Run	Read-only. No hazards.
R_ScbPerfBuckets_Event	Takes effect the next time the specific bucket starts or resumes counting.
R_ScbPerfBuckets_IfOther	Takes effect immediately. If this bucket is the one being counted, the count currently being calculated may spuriously count or lose a few events. Not recommended.
R_ScbPerfBuckets_Hist	Takes effect immediately. If this bucket is the one being counted, the count currently being calculated may spuriously count or lose a few events. Not recommended.
R_ScbPerfCounts_Count	If this bucket is not the one being counted, the value will remain. If this bucket is the one being counted, the new count may be used, or the value may be overwritten with the pre-written value plus the count from the current interval.

10.11 Connecting to SCBS

10.11.1 List of Slaves

The ICE9 has slaves across most of the chip. A complete list of slaves is listed in the AddrSubId enumeration in 16.6.6. Any row with a clock specified in the Clk column includes a Scb slave.

10.11.2 Slave I/O Transactions

Slaves connect their I/O registers to the SCBS using a simple request/acknowledge interface, with only one transaction ever outstanding. On a single cycle pulse of the scbs_x_req_r line, user logic decodes the address, write-not-read signal, and write address if applicable. When the user logic has completed the operation, it drives read data if applicable and pulses the x_scbs_ack_r line. The scb_ack_r must be pulsed after every scb_req_r, even if the address does not correspond to any valid register address. Additionally, invalid read addresses should return 0.

10.11.3 Slave Performance Counting Interface

Each slave uses the scbs_x_eventId#_xr signals to select which event is to be counted. The event is returned to the Scb slave counter as a single bit. The lowest cost way for user logic to implement this is probably a combination

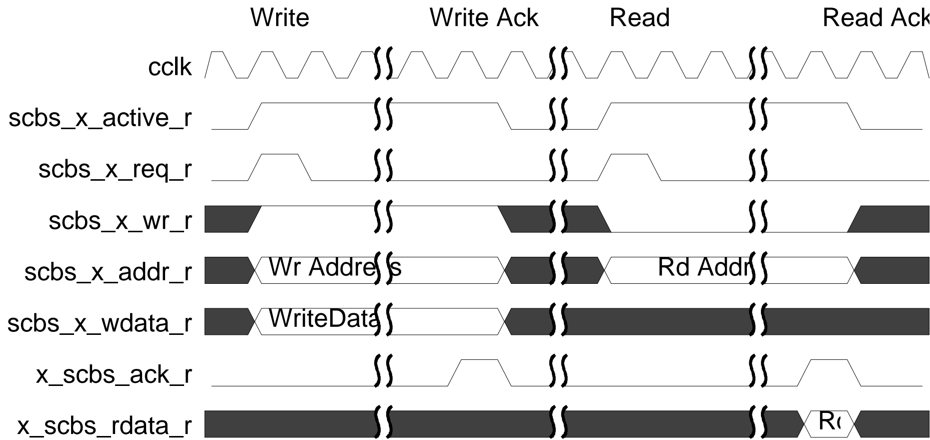


Figure 10.3: SCB Slave Timing

of muxes and AND gates which return a 0 whenever the address doesn't match the desired event. A tree of these in each sub-block then feed a reduction OR tree, or see 10.1. Up to 8 flops may be introduced by the user logic at any point in this computation, as the SCB will discard the earliest sampling cycles.

If any slave has additional registers related to performance counting, those registers should be in a unique 64KB page to allow software to map it into user virtual address space.

Algorithm 10.1 SCB User Event Counting Example

```
always @ (posedge clk) begin
   if (scbs_cpu_active_pr) begin // Clock gate
      m_scbs_event_p <= {eventMux(scbs_cpu_eventId1_pr),
                         eventMux(scbs_cpu_eventId0_pr)};
   end
end
function eventMux;
   input [7:0] select;
   case (select)
    `E_XxxScbEvent_CYCLES: eventMux = 1'b1;
    `E_XxxScbEvent_DCHIT:  eventMux = (signal_high_on_DC_hit);
     default:              eventMux = 1'b0;
   endcase
endfunction
```

10.12 SCB Internals

This section describes the SCB internals.

10.12.1 PMI Interface

The connection between the PMI and the SCB master is a simple pulsed request/ acknowledge handshake. The request and acknowledge handshake is nearly identical to the slave interface, with the addition of the upper address bits. See Figure 10.3.

10.12.2 SCB Bus Protocol

All slaves on a particular SCB bus all operate on the same clock domain; additional chains are used for each unique clock domain. Thus there are multiple SCB chains on the chip; presently for the pclk, cclk, d0clk, d1clk,

and sclk. (We can never have a iclk chain, as the iclk is not running in non-PCI connected chips.) The master contains the synchronizer flops between the cclk (Scb master's domain) and the slave bus's domain. This places all of the synchronizers in one place, and is more logic efficient then requiring each slave to have a synchronizer.

Each SCB bus consists of 3 wires connected in a chain, plus the clock. The 3 bits of the data bus consist of two logically seperate signals, the command and data bits, that are bussed simply so the top level interconnect need only concern itself with a single bus.

In/Out	Signal Name	Definition
in	scb_chaini_dat[1:0]	Data input.
in	scb_chaini_dat[2]	Command input.
out	scb_chaino_dat[1:0]	Data output.
out	scb_chaino_dat[2]	Command output.

10.12.3 ICE9 Bit Sequence

Every clock cycle, data is present on the _dat wires. A shift sequence begins with with a start bit on _dat[2], and proceeds from MSB to LSB. The _dat[2] input feeds a 17 bit command shift register. Likewise, _dat[0] feeds the even bits of a 32 bit data register, and _dat[1] feeds the odd bits of the 34 bit data register. The bits of the command and data registers are allocated as follows:

Register	Valid during what commands?	Definition
Cmd[16]	All	Start bit.
Cmd[15:12]	All	Command (see ScbCmd encoding.)
Cmd[11:2]	Read,Write	Address [11:2].
Cmd[1]	-	Reserved.
Cmd[0]	All	Match bit. Set by slave when command detected with an address matching a slave's address.
Data[33:32]	All	Indicates what acknowledgements are present on the data bus. See ScbDataAck encodings.
Data[31:0]	Read, Write	Data
Data[30:2]	AddrH	Address [30:2].
Data[30:24]	Count	Slave number for Event ID 1.
Data[23:16]	Count	Event ID 1.
Data[14:8]	Count	Slave number for Event ID 0.
Data[7:0]	Count	Event ID 0.

10.12.4 TWC9+ Bit Sequence

TWC9A changes the protocol slightly to allow 64 bit slaves. It also still supports 32 bit slaves without forcing them to implement a fill 64 bit shifter, by insuring the first 32 shifts (with bits 64:33) can simply be dropped by 32 bit slaves and still have everything work out.

Every clock cycle, data is present on the _dat wires. A shift sequence begins with with a start bit on _dat[2], and proceeds from MSB to LSB. The _dat[2] input feeds a 33 bit command shift register. Likewise, _dat[0] feeds the even bits of a 66 bit data register, and _dat[1] feeds the odd bits of the 66 bit data register. The bits of the command and data registers are allocated as follows:

Register	What commands?	Definition
Cmd[32]	All	Start bit.
Cmd[31:17]	All	Reserved. Note 32 bit slaves shift past these bits and so cannot decode them.
Cmd[16]	All	Finished shift bit. 32-bit slaves complete command shifting with this bit in what would normally be the start bit. Therefore, the master sets this bit so the code may assert the shifting was properly completed. This bit may be stolen for other purposes if the assertions are removed.
Cmd[15:12]	All	Command (see ScbCmd encoding.)
Cmd[11:2]	Read,Write	Address [11:2].
Cmd[1]	Read, Write	Double-word access. If a 32 bit slave sees this set, it's an error.
Cmd[0]	All	Match bit. Set by a slave when command detected with an address matching the slave's address.
Data[65:64]	All	Indicates what acknowledgements are present on the data bus. Slaves do not decode these bits. See ScbDataAck encodings.
Data[63:0]	Read, Write	Data. 32-bit accesses have the data replicated on both the upper and lower words.
Data[63:32]	AddrH	Reserved. Note 32 bit slaves shift past these bits and so cannot decode them.
Data[30:2]	AddrH	Address [30:2].
Data[1:0]	AddrH	Reserved.
Data[63:32]	Count	Reserved. Note 32 bit slaves shift past these bits and so cannot decode them.
Data[30:24]	Count	Slave number for Event ID 1.
Data[23:16]	Count	Event ID 1.
Data[14:8]	Count	Slave number for Event ID 0.
Data[7:0]	Count	Event ID 0.

10.12.5 Commands

The 4 bit command, enumerated in 10.14.3 decodes to the following operations:

10.12.5.1 Idle

The Idle operation is used during bus idle, and is ignored by all slaves.

10.12.5.2 Reset

The Reset op causes SCB slaves to clear the slave's internal internal state, and is reached by continuously sending all ones on the _dat[0] input. Reset persists until a pure Idle (all zeros) is received. This allows slaves to be reset on a hang without losing register state.

10.12.5.3 AddrH

The AddrH op causes the last 32 bits of data shifted over _dat[1:0] to be loaded into the high bits of the address register.

10.12.5.4 Write

The Write op loads the low I/O address from the low 4 bits of the command shifter, and asserts a write request to the SCB user logic. When the SCB user logic accepts the write with ack_r, the slave passes the acknowledgement

to the SCB master by shifting a single pulse onto the _dat[1] output.

10.12.5.5 Read

The Read op loads the low I/O address from the low 4 bits of the command shifter, and asserts a read request to the SCB user logic. When the SCB user logic has the read data ready, it returns it to the SCB slave with an acknowledge. The slave acknowledges the Read to the SCB Master with a start pulse on the _dat output of _dat[2:0]=3'b011, followed by 16 double-bits of read data.

10.12.5.6 Count

The Count op causes the high bits of the address to be compared to the slave's write data register, and if matching, event data to be muxed onto the _dat[1:0] outputs. The two events being counted may come from different slaves, so two slave numbers are sent along with the Count op; either one matching will drive the appropriate _dat lines. Counting is "sticky" in that after the state machine returns to idle, it continues counting until the next _dat[2] start bit.

10.13 Chip Reset

On chip reset, all SCB master registers (except RAM) are cleared and counting is disabled. Software needs to clear the RAM by writing zeros to it during boot.

During a SCB user driving the reset line into the SCB slave, that slave will ignore all SCB transactions, and that slave places its SCB bus is in bypass mode. This allows each slave to have a different reset, and all other slaves not in reset to still be programmable via the SCB. However, any slave's reset must be deasserted only while the SCB bus is idle, to avoid decoding the first command incorrectly.

Also during SCB user reset, a SCB slave will drive zeros on the write data wires. This allows SCB slaves to OR their CSR write enables with reset, so they will load the data bus and thus the zeros on reset. (Registers which affect the pins still need async reset, however.) This is more space and power efficient then using (a)synchronous resets on every data bit of every control register.

10.14 Registers and Definitions

10.14.1 Package Attributes

Package

chip_scb_spec

Attributes

-public_rdwr_accessors

10.14.2 Definitions

Defines

SCB

Constant	Mnemonic	Definition
32'd32	DATAWIDTH	Data Bus Width. Default width of data bus in bits.
32'd7	SLAVEBITS	Bits of address for unit number. Number of upper address bits that correspond to choosing which SCB Slave will be addressed.
32'd8	COUNTBITS	Bits of counter events. Number of lower address bits used per-slave for counting events.
32'd60	DELAY_NS_BC	Speed register delay, best conditions. Nanoseconds.
32'd95	DELAY_NS_TYP	Speed register delay, typical conditions. Nanoseconds.
32'd190	DELAY_NS_WC	Speed register delay, worst conditions. Nanoseconds.

10.14.3 Command Enumerations

ScbCmd specifies the bit encodings for the commands encoded in the first bits sent over the serial bus.

Enum

ScbCmd

Constant	Mnemonic	Definition
4'b0000	IDLE	Idle.
4'b0001	ADDRH	Latch High Address.
4'b001x		Reserved
4'b01xx		Reserved
4'b1000		Reserved
4'b1001	WRITE	IO Write.
4'b1010	READ	IO Read.
4'b1011	COUNT	Event Count.
4'b110x		Reserved
4'b1110		Reserved
4'b1111	RESET	Reset SCB slave state.

10.14.4 Data Ack Enumerations

ScbDataAck specifies the bit encodings for the high two data bits. In addition for slave transactions, the MSB is the start bit, so must be set.

Enum

ScbDataAck

Constant	Mnemonic	Definition (if from Slave)	(Definition if from Master)
2'b00	NONE	NA - No start bit	AddrH - No acks needed
2'b01	NEED	Read data 64 bit ack	Need later acknowledgement from slave.
2'b10	WRITE	Write accepted	First bit of write passed around loop, or last bit of count passed through loop
2'b11	READ	Read data 32 bit ack	NA

10.14.5 SCB Performance Events

The following SCB internal events are trackable by SCB statistical event counting.

Enum

ScbScbEvent

Attributes

-descfunc

Constant	Mnemonic	Definition
8'h00	CYCLES	Core clock cycles. Always counts.
8'h01	CYCLES_D2	Internal verification only. Repeats high for 2 cycles, then low for 2.
8'h02	MAGIC0	Internal verification only. Counts cycles where R_ScbPerfCtl_MagicEvent[0] is true.
8'h03	MAGIC1	Internal verification only. Counts cycles where R_ScbPerfCtl_MagicEvent[1] is true.
8'h04-8'hff		Reserved. Returns zero.

10.14.6 Chip Revision Register

Register

R_ScbChipRev

Attributes

-kernel

Address

0xE_0800_0000

Bit	Mnemonic	Access	Reset	Type	Definition
31:16	Features	R	pins		Feature bit. Bits in this region will be allocated to indicate optional features or enhancements, as they are specified. Overlaps allowed. Bit 16 is 1 in ICE9A1 so we can determine proper mask selection.
15:8	Product	R	pins	AddrProduct	Chip Product/Revision. Revision number of the chip product, returns ICE9, ICE9B, etc; incremented for each new major product. Use AddrProduct enumeration for exact values, see 16.6.4 on page 846.
7:0	Rev	R	pins		Minor Chip Revision. Revision number of the chip, bumped for different silicon passes or minor releases. This is metal-mask programmable.

10.14.7 Chip Number Register

Register

R_ScbChipNum

Attributes

-kernel

Address

0xE_0800_0008

Bit	Mnemonic	Access	Reset	Type	Definition
31:16					Reserved
15:11	System	RW	0		System number. Reserved for future use; written by module service processor, and read by software.
10:5	Board	RW	0	(MspSlotId)	Slot ID number. Intended to be written over SysChain by module service processor, and read by software. Identical to MSP GPIO slot ID enumeration.
4:0	Node	R	pins		Chip number on board (0-26). Hardcoded value from sys_node[4:0] input pins.

10.14.8 Chip Null Subcomponent Register

This register is used for simulation purposes only. In real hardware it always returns 0.

Register

R_ScbChipMissing

Attributes

-kernel

Address

0xE_0800_0010

Bit	Mnemonic	Access	Reset	Type	Definition
31	Cached	R	0		Value Cached. Used in C code to indicate register value has been cached.
30:12					Reserved
11	Uart	R	pins		Model has no Uart function. Verification use only, 0 on HW.
10	Scb	R	pins		Model has no Scb Master function. Verification use only, 0 on HW.
9	Prc	R	pins		Model has no Prc function. Verification use only, 0 on HW.
8	Ocla	R	pins		Model has no Ocla function. Verification use only, 0 on HW.
7	I2c	R	pins		Model has no I2c function. Verification use only, 0 on HW.
6	Fsw	R	pins		Model has no Fsw function. Verification use only, 0 on HW.
5	Fl	R	pins		Model has no Fl function. Verification use only, 0 on HW.
4	Dma	R	pins		Model has no Dma function. Verification use only, 0 on HW.
3	Ddr	R	pins		Model has no Ddre/Ddro functions. Verification use only, 0 on HW.
2	Coh	R	pins		Model has no Cac or Coh functions. Verification use only, 0 on HW.
1	Cpu15	R	pins		Model has no Cpu1-CpuN functions. Verification use only, 0 on HW.
0	Cpu0	R	pins		Model has no Cpu0 function. Verification use only, 0 on HW.

10.14.9 Chip Speed Register

R_ScbSpeed is used to determine the latency through a delay line to provide a very rough approximation of the speed of the part. Software hits the GO bit, then waits for the GO bit to clear. It then reads the Count value. This experiment must always be done in pairs: The first will measure one edge transition (say rising-to-falling), the second will measure the opposite transition. The numbers will differ by 15% or so. Both numbers should be reported.

Register

R_ScbSpeed

Address

0xE_0800_0020

Bit	Mnemonic	Access	Reset	Type	Definition
31	Go	RW1CS	0		Go. When written with one, set GO bit and start counting. After the delay is calculated, the go bit will clear and the new count will be visible.
30:10					Reserved.
9:0	Count	R	0		Delay line time. After Go completes, number of pclk cycles plus 2 taken to count a delay line of SCB_DELAY_NS_TYP ns. See the note about double measurements in the beginning of this section.

10.14.10 General Purpose IO Register

Register

R_ScbGpio

Address

0xE_0800_0040

Bit	Mnemonic	Access	Reset	Type	Definition
31:20					Reserved.
19:16	inp	R	pins		GPIO input data. This may not match the output data when oe is asserted if a stronger driver is present on the input pin. Bit 0 reads value on sys_gpio (spare) input pin. Bits 3:1 reserved for future use.
11:8	oe	RW	0		GPIO output enable. If bit 0 set, drive sys_gpio (spare) pin with _data value. If clear, tristate. Bits 3:1 reserved for future use.
3:0	data	RW	0		GPIO output data. Bit 0 value is driven to sys_gpio (spare) pin if _oe is set. Bits 3:1 reserved for future use.

10.14.11 LED Register

Register

R_ScbLed

Address

0xE_0800_0048

Bit	Mnemonic	Access	Reset	Type	Definition
31:1					Reserved.
0	led	RW	0		LED status. If set, assert sys_led_l pin by enabling its open drain driver, pulling sys_led_l low. If not set, sys_led_l is hi-impedance.

10.14.12 Attention Chip Register

With the associated R_ScbAtnMsp register, the attention chip register provides a signaling interface between the Chip and MSP.

R_ScbAtnChip forms a MSP to/from Chip communication channel in conjunction with the R_SysTapAtnMsp register in 12.6.15.

To send data to the MSP, the chip implements the code in 10.2.

Algorithm 10.2 R_ScbAtnChip algorithm

```
send_something() {
    do { rdata = read_of(R_ScbAtnChip);
       } while (rdata & bit(SendVld)));
    write_of(R_ScbAtnChip,
             bit(SendVld) | send_data);
}
receive_something() {
    rdata = read_of(R_ScbAtnChip);
    if (rdata & bit(RecvVld)) {
        write_of(R_ScbAtnChip, bit(RecvTaken));
        // process data in rdata
    }
    // Else nothing to receive
}
// Better code could both send and receive data simultaniously.
```

Register

R_ScbAtnChip

Attributes

-kernel -writeonemixed

Address

0xE_0800_0060

Bit	Mnemonic	Access	Reset	Type	Product	Definition
31						Reserved.
30					**ICE9A**	Reserved.
30	TxIntMask	RW	0		**ICE9B+**	Transmitter Empty Interrupt Enable. Indicates chip interrupt should be asserted if _SendVld is clear and more data may be sent. If clear, no interrupt. Note transmitter empty is the idle-state condition, so this bit should never be left on once all data is sent. First implemented in ICE9B.
29	RecvInt	RW	0			Receiver Ready Interrupt Enable. Indicates chip interrupt should be asserted if _RecvVld is also asserted. If clear, no interrupt.
28	RecvTaken	W1C	0			Receive Data Taken. Write one to send to module processor indication that RecvData was accepted, and clear _RcvVld.
27	RecvVld	R	0			Receive Data Valid. Valid flag from module processor, identical to R_ScbAtnMsp_SendVld. Indicates module processor data is valid to be read from RecvData. When data is accepted, chip writes _SendTaken.
26	SendVld	RW1CS	0			Send Data Valid. Write one to set and indicate new send data for MSP. Cleared when MSP takes the data.
25:0	RecvData	R	0			Receive Data. Overlaps SendData. If RecvVld is set, returns the next data to be received from the MSP. Note this is different data then that written.
25:0	SendData	W	0			Send Data. Overlaps RecvData. If SendVld is simultaniously being written with a one, enqueues new send data for the MSP, and sets SendVld.

10.15 Debug Attention Interrupt Register

The ScbAtnInt register is used by the MSP to select what should assert the attention signal.

This register should only be written by the MSP. (It would be a SysChain register, but leaving it in SCB space saves a significant number of syncronizer flops, as it must reside on a clock which is always running.)

Register

R_ScbAtnInt

Attributes

-Product=ICE9B+

Address

0xE_0800_0070

Bit	Mnemonic	Access	Reset	Type	Product	Definition
31	Atn	R	X		**ICE9B+**	Attention Asserted. True if the sys_atn pin is asserted, IE if any request bit is asserted and the corresponding mask is asserted.
30	NonComAtn	R	X		**ICE9B+**	Non-Communication Attention Asserted. True if anything other then _RxAtn or _TxAtn is asserting attention. This bit is duplicated in R_SysTapAtnMsp_NonComAtn to reduce polling in the MSP fast path.
29					**ICE9B+**	Reserved.
28:25	Cpu6DMMask	RW	0		**TWC9A+**	CPU9:6 Debug Mode Mask. See _CpuDMMask.
24	TxAtnMask	R	0		**ICE9B+**	Transmit Empty Mask. This is a read only copy of R_SysTapAtnMsp_TxAtnMask; use that register to enable/disable transmit interrupts.
23	RxAtnMask	RW	0		**ICE9B+**	Receive Ready Mask. Enables _AtnRx asserting attention.
22	OclaDMMask	RW	0		**ICE9B+**	OCLA Debug Mode Mask. Enables _OclaDM asserting attention.
21:16	CpuDMMask	RW	0		**ICE9B+**	CPU5:0 Debug Mode Mask. Enables corresponding _CpuDM asserting attention. Note bits for CPU6-9 are not contiguous, see the _Cpu6DMMask field.
15:13					**ICE9B+**	Reserved.
12:9	Cpu6DM	R	X		**TWC9A+**	CPU9:6 in Debug Mode. See _CpuDM.
8	TxAtn	R	1		**ICE9B+**	Transmit Empty. R_SysTapAtnMsp_SendVld is clear, indicating more data may be transmitted.
7	RxAtn	R	0		**ICE9B+**	Receiver Ready. R_SysTapAtnMsp_RecvVld is set, indicating data is ready to be received.
6	OclaDM	R	0		**ICE9B+**	OCLA Requesting Debug Mode. Asserted when the OCLA is requesting a Debug Interrupt; identical to R_ScbDInt_OclaDM.
5:0	CpuDM	R	X		**ICE9B+**	CPU5:0 in Debug Mode. Asserted when the corresponding CPU is in Debug Mode; identical to R_ScbDInt_CpuDM. Note bits for CPU6-9 are not contiguous, see the _Cpu6DM field.

10.16 Debug Interrupt Register

Register

R_ScbDInt

Attributes

-Product=ICE9B+ -noregtestcpu

Address

0xE_0800_0078

Bit	Mnemonic	Access	Reset	Type	Product	Definition
31:28					**ICE9B+**	Reserved.
27:24	Cpu6DM	R	X		**TWC9A+**	CPU9:6 in Debug Mode. See _CpuDM.
23:20					**ICE9B+**	Reserved.
19:16	SendDInt6	RW	0		**TWC9A+**	Send CPU9:6 a Debug Interrupt. See _SendDInt.
15	OclaToAll	RW	0		**ICE9B+**	OCLA causes CPU Debug Interrupt. If set, when _OclaDM asserts, assert DINT to all CPUs.
14	CpuToAll	RW	0		**ICE9B+**	CPU Debug Mode causes CPU Debug Interrupt. If set, when any CPU enters debug mode and _CpuDM asserts, assert DINT to all CPUs. Thus when one CPU takes a debug execption, they all will.
13:8	SendDInt	RW	0		**ICE9B+**	Send CPU5:0 a Debug Interrupt. Set high to assert DINT to the specified CPU. (Note DINT is edge sensitive at the CPU.) After setting, poll on this register until the corresponding _CpuDM bit asserts, then clear this bit. Note CPUs 6-9 are not contiguous.
7					**ICE9B+**	Reserved.
6	OclaDM	R	0		**ICE9B+**	OCLA Requesting Debug Mode. Asserted when the OCLA is requesting a Debug Interrupt.
5:0	CpuDM	R	X		**ICE9B+**	CPU5:0 in Debug Mode. Asserted when the corresponding CPU is in Debug Mode. Note CPUs 6-9 are not contiguous.

10.17 Performance Counting Registers

10.17.1 Interrupt Register

Register

R_ScbInt

Attributes

-kernel

Address

0xE_0800_0080

Bit	Mnemonic	Access	Reset	Type	Product	Definition
31	Irq	RO	0			Interrupt asserted. Asserted to represent the interrupt output, namely whenever the given interrupt bit is on in this register, and the interrupt mask is enabled for that bit.
30:3						Reserved.
2					ICE9A	Reserved.
2	AtnTxInt	R	1		ICE9B+	Attention Transmit Empty Interrupt. More data may be sent to R_ScbAtnChip. Send data to clear the interrupt. Note this bit resets to 1, as after reset the send buffer is empty and ready to transmit.
1	AtnInt	R	0			Attention Interrupt. Data is ready in R_ScbAtnChip. Accept the data to clear the interrupt.
0	PerfInt	RW1C	0			Performance Interrupt. A counter has overflowed R_ScbPerfCtl_IntBit. Write 1 to clear. R_ScbIntReq_PerfInt can be written to assert this interrupt.

10.17.2 Interrupt Mask Register

Register

R_ScbIntMask

Attributes

-kernel

Address

0xE_0800_0088

Bit	Mnemonic	Access	Reset	Type	Definition
31:3					Reserved.
2					Reserved. (Attention transmit empty interrupts are maskable via the R_ScbAtnChip_TxInt register.)
1					Reserved. (Attention Interrups are maskable via the R_ScbAtnChip_Int register.)
0	PerfInt	RW	0		Performance interrupt mask. Enables R_ScbInt_PerfInt asserting an interrupt.

10.17.3 Interrupt Request Register

Register

R_ScbIntReq

Attributes

-kernel

Address

0xE_0800_0090

Bit	Mnemonic	Access	Reset	Type	Definition
31:3					Reserved.
2					Reserved. (Attention transmit empty interrupt can be created with the R_ScbAtnChip register.)
1					Reserved. (Attention Interrups can only be requested by the MSP.)
0	PerfInt	W1CS	0		Performance interrupt request. Asserts R_ScbInt_PerfInt.

10.17.4 Performance Control Register

Register

R_ScbPerfCtl

Attributes

-kernel

Address

0xE_0801_0000

Bit	Mnemonic	Access	Reset	Type	Definition
31:13					Reserved.
12:11	MagicEvent	RW	0		Model Magic events. For verification, allow creating of raw events trackable with ScbScbEvent_MAGIC0 and _MAGIC1.
10	AddrAssert	RW	1		Model Magic address assertion. Fire an assertion on a read or write to a bad address. No function in silicon; reads to bad addresses always return 0xFFFFFFFF regardless of this bit.
9	NoInc	RW	0		Disable automatically incrementing the bucket. When clear, after each _Interval, increment R_ScbPerlBuckNum register. When set, always use the specified static R_ScbPerlBuckNum.
8:4	IntBit	RW	31		Interrupt bit select. Bit number, that when gets set asserts an interrupt. Thus the default of 31 will interrupt before a counter may overflow, and a value of 0 will interrupt when any event occurs (bit 0 asserts). Interrupts occur when the the count bit overflows, and don't wait until the interval completes. Interrupts do not stop the counting.
3:0	Interval	RW	3		Sampling interval. Log2 number of cycles to spend on sampling each bucket. 0=32 cycles, 1=64 cycles, ..., 15=1M cycles. Note setting a 1M cycle interval will require nearly a second before the entire RAM is sampled, which will delay R_ScbPerfStat_Run clearing by up to a second.

10.17.5 Performance Histogram Register

Register

R_ScbPerfHist

Attributes

-kernel

Address

0xE_0801_0008

Bit	Mnemonic	Access	Reset	Type	Definition
31:20					Reserved.
19:0	HistGte	RW	1		Histogram greater-than-equal value. Running experiments counting a "waiting-for" type of event, and varying HistGte, will give enough data to generate a histogram of latency versus probability. For each bucket, if R_ScbPerfBuckets_Hist is cleared, this register is ignored and that bucket counts cycles. If R_ScbPerfBuckets_Hist is set, and _HistGte == 0 gives unspecified results. (As it is meaningless to look for the times just a 0 to 0 transition occurs.) If R_ScbPerfBuckets_Hist is set, and _HistGte == 1, the bucket counts the number of occurances of the serial regular expression 0+1+, which is simply the number of positive edges. If R_ScbPerfBuckets_Hist is set, and _HistGte >= 2, count one for ever time the event is high for >= R_ScbPerfHist's number of cycles. I.E. With _HistGte=2, count 0+11+. With _HistGte=3, count 0+111+, etc. If R_ScbPerfBuckets_Hist is set, and _HistGte == all ones gives unspecified results.

10.17.6 Performance Bucket Number Register

Register

R_ScbPerfBuckNum

Attributes

-kernel

Address

0xE_0801_0010

Bit	Mnemonic	Access	Reset	Type	Definition
31:27					Reserved.
15:8					Reserved. (for increasing number of buckets.)
7:0	Bucket	RW	0		Bucket number. The current bucket being sampled. This will automatically increment by 2 if counting is in progress and R_ScbPerfCtl_NoInc is clear. Bit 0 is ignored, as counting is always done in bucket pairs.

10.17.7 Performance Enable Register

Register

R_ScbPerfEna

Attributes

-kernel

Address

0xE_0801_0020

Bit	Mnemonic	Access	Reset	Type	Definition
31:1					Reserved.
0	ena	RWSL	0		Enable sampling. Write one to start sampling/counting. Counting will continue as long as this remains set. Clear to end counting at next opportunity: when interval completes on the last bucket or R_ScbPerfCtl_NoInc and any bucket. R_ScbPerfStat_Run will clear when the final sample is completed.

10.17.8 Performance Status Register

Register

R_ScbPerfStat

Attributes

-kernel

Address

0xE_0801_0028

Bit	Mnemonic	Access	Reset	Type	Definition
31:1					Reserved.
0	run	R	0		Sampling is running. True when counting is active. The count ram will not have the most recent counts until this deasserts.

10.17.9 Performance Bucket Configuration

The R_ScbPerfBuckets registers contains the event number and controls for when the associated bucket is counted.

Register

R_ScbPerfBuckets[255:0]

Attributes

-noregtestcpu_reset -kernel

Address

0xE_0801_4000-0xE_0801_43FC

Bit	Mnemonic	Access	Reset	Type	Definition
31:18					Reserved.

17:16	ifOther	RW	FW0		Count if AND other event. 00 or 11, normal operation. When 01, only increment the count in those cycles where this event and the opposite bucket's (odd bucket's event for even buckets, even bucket's event for odd buckets) raw event before applying ifOther or histogramming is asserted. When 10, only count when the event AND NOT the opposite event. Note this only works when comparing against other events in the same clock domain. (See 16.6.6 for the clock domain list, and note IfOther counting for two events in the same subchip ID is always ok.)
15	hist	RW	FW0		Histogram or count edges on the specified event. Otherwise if clear, count cycles where the event is asserted. See R_ScbPerfHist. This detection occurs after the ifOther equation.
14:0	event	RW	FW0		Event ID to count. Consists of the SCB slave number (see 16.6.6), concatenated with the 8-bit event number inside that slave. Also see the AllEvent enumeration (in source only, not a spec). Events not specified return zero counts. - In Twc9a and followons, if both pairs of events contain the special value AllEvent_INVALID (with encoding 0), this pair will not be sampled, and sampling will quickly continue to the next bucket. - For a list of AllEvent (or Ice9_AllEvent) enumerations with descriptions, see "<project>/sw/include/sicortex/ice9/ice9_all_spec_sw.h". These enumerations provide you all 15 bits for the "event" field of R_ScbPerfBuckets. Note that these enumerations don't list the OCLA events that are available to count. See the On Chip Logic Analyzer chapter. In OCLA LAC, all the trigger-block triggers are available, after delays have are applied. In OCLA TRBCs, the outgoing triggers are available (like getting them from LAC but without delays). In OCLA TRBVs, the 32 incoming data signals are available.

10.17.10 Performance Count Ram

The ScbPerfCounts registers contain the counts for each bucket, indexed by bucket number.

Register

R_ScbPerfCounts[255:0]

Attributes

-noregtestcpu_reset -kernel

Address

0xE_0801_8000-0xE_0801_83FC

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	count	RW	FW0		Performance counts. Number of cycles for which the given bucket's event was asserted. For the read to include the most recent interval's results, R_ScbPerfStat_run must be clear.

Chapter 11

On Chip Logic Analyzer

[$Id: chipocla.lyx 50693 2008-02-07 16:01:46Z wsnyder $]

11.1 Overview

The On-Chip Logic Analyzer (OCLA) provides debug capabilities for the processor segments and their associated L2 caches (PSX), the fabric switch (FSW), the DMA engine, the two coherence units (COHE and COHO) and the PMI unit. The OCLA is distributed around the chip and includes Capture Trace Blocks (CTBs), Trigger Blocks (TRBs), and a central controller (LAC). The trigger blocks come in two varieties; Codeword Trigger Blocks (TRBCs) and Vector Trigger Blocks (TRBVs). Some CTBs and TRBs have muxed inputs to allow larger numbers of signals to be sampled or triggered upon on a mutually exclusive basis. The CTBs, TRBs, and LAC are accessed via the Serial Configuration Bus (SCB). The module service processor may access the OCLA via the SCB hook on the SysChain.

11.2 Differences, Bugs, and Enhancements

11.2.1 Product and Chip Pass Differences

1. ICE9B fixes GO->0 should shut OFF collection, bug2246. CollectTrace can be left ON by stopping an OCLA program that had not yet seen it's trigger. CollectTrace can only be controlled by a running OCLA program, so you can't shut it off by SCB writes. While CollectTrace is ON, you cannot dump any CTBs. Workarounds: (a) A Diagnostics Dash script has been written that loads and runs a minimal OCLA program to shut off CollectTrace. (b) The OCLA dump program has been written to detect CollectTrace=ON, and exit with meaningful error message. (c) OCLA Dash scripts and all example OCLA programs have been written with a âgraceful exitâ option, where a specific register-write tells it to shut CollectTrace OFF and stop watching for the trigger it didn't get yet.

2. ICE9B adds new INCRBTH Opcode, bug2179. In ICE9A, although OCLA has 2 counters, you cannot count 2 events concurrently, because if both happen on same clock there's no way to increment both counters.

3. ICE9B enlarges counters from 12 to 16 bits, bug2244.

4. ICE9B fixes PMI CTB ExtMuxSel wired to TRBC, bug1959. The ExtMuxSel wires of OCLA PMI CTB were wired to the SCB register that's supposed to control OCLA PMII TRBC. To workaround, write desired PMI CTB ExtMuxSel value to ExtMuxSel field in control register for PMII TRBC. Fortunately, PMII TRBC has no ExtMux, so this field is otherwise unused. Simplest solution without determining whether you have Ice9A or ICE9B is write desired PMI CTB ExtMuxSel value to both ExtMuxSel fields.

5. ICE9B fixes CAC trigger PrbState obscured by WtPrb2L2, bug1995. OCLA CAC TRBC mux=2 signals PrbState[2:0] had WtPrb2L2 OR-ed into PrbState[2]. To workaround, don't use PrbState as a trigger, or only trigger on PrbState groups of state that you can identify with bits [1:0].

6. ICE9B fixes CAC trigger W0Hit/W1Hit instead of W0Miss/W1Miss, bug2243. In ICE9A, both CAC Trigger Block and Collector Block hookups: (a) Change W0Miss/W1Miss to something better, perhaps

W0Hit/W1Hit. Miss is including Idle and I/O. (b) Adjust flops so W0Hit/W1Hit in same clock with related signals. To workaround, (a) qualify with not-Idle and not-IO. (b) Separately feed Hit and the other signals to LAC in separate triggers, then align them with Dly regs in LAC.

7. MIGHTFIX: TWC9A might fix OCLA to SCB uses LAC triggers, bug1717. Passing OCLA events from trigger blocks to SCB Counters ties up LAC trigger configuration, usually preventing simultaneous OCLA use for other purposes. To workaround, accept that you are tying up OCLA with this. The cross connections between OCLA and SCB counting may not be used that much. You might prefer to count SCB events in SCB counters, and count OCLA events in OCLA counters.
8. MIGHTFIX: TWC9A might allow trigger delays for blocks located in other than the CCLK domain, bug1854.
9. MIGHTFIX: TWC9A might add capture mux settings for the CPU program counter and L2<->L1 signals.
10. NEED IMPL: TWC9A might add capture mux settings for the FSW links 1 and 2, bug2232.
11. MIGHTFIX: TWC9A might fix DMA CTB qualifier in wrong clock, bug2193. In DMA's hookups to OCLA, the ue_xxx_DbgValid_c2a signal is sent into the trigger block and CTB, when really it should be delayed by two more cycles. In the CTB as a qualifier we pretty much cannot use it, because you want to use it in combination with other signals like DbgThread_c4a and DbgPc_c4a. To workaround, only do un-qualified collection in DMA CTB. In DMA trigger block, send it and other signals separately on the 2 triggers to LAC, where the Dly regs can align them.
12. MIGHTFIX: TWC9A might add a WtAddr sticky overflow bit, bug2207.
13. MIGHTFIX: TWC9A

11.2.2 Known Bugs

1. Overflow bits still set as OCLA starts, bug1825. OCLA's automatic clearing of counter overflow bits when you start LAC program is delayed a clock or two. Early instructions in LAC program can falsely trigger on overflow depending on the previous use of OCLA. To workaround, never branch on Counter Overflows in first 2 instructions of any LAC program.
2. C CTB WtAddrClr triggered by any address in CTB, bug2026. Writing 0x10 to any SCB register address in a particular Ocla CTB can trigger WtAddrClr (clear write address reg). This even includes unused addresses within the SCB address space of a CTB. To workaround, never write any of the read-only registers.

11.2.3 Possible Enhancements

1. Make both LAC counters 32-bit (currently 16-bits plus sticky overflow bit). There's only one instance of the LAC, so this is very affordable. We've wanted bigger counters when writing LAC programs, and unanticipated but valuable use of OCLA as a highly-configurable counter would benefit from full 32-bit counters.
2. Separate "GO" Register. When you write OCLA management software for one of Ice9's embedded processors, or for the external SSP, you tend to write one function that configures OCLA ahead of time, and another function to tell OCLA to "GO" at roughly the right moment. Currently the GO bit shares register R_LacCtl with some configuration fields that need to be written correctly for what you want OCLA to do. This contributes to messy software design in that you must have handy the values to write to those fields when you write a 1 to GO to start the LAC program. It would be nice if all OCLA configuration could be encapsulated in, and completed by an OCLA configuration function.
3. If SCB reg addresses are cheap, consider breaking R_LacCtl into 3 or 4 registers by type of access, making software easier to write.
4. Collect ON/OFF by Register Write. Provide a super-simple alternative to writing a LAC program, for when exact timing of collection is not critical. Provide one or two registers that allow you, by SCB register write alone, to turn on and off CollectTrace to the CTBs. This allows someone with minimal knowledge of OLCA to quickly collect some trace information and read it out, just by doing easy-to-understand SCB writes and reads. Some semi-steady-state activities can be viewed at an arbitrary time, or you could try more than once till you see it. Or, for more accuracy, you could have Ice9 embedded processor code trigger collection at

roughly the right time, and rely on the 1024-entry size of the CTBs to give you a pretty big window to land in. These reg writes would the same logic as the SETCOLL and CLRCOLL opcodes from LAC.

5. Trigger by Register Write. There are ways to do this now, but they're a little obscure. I'm suggesting a very-simple up-front way to trigger your LAC program by writing an SCB register in LAC who's sole purpose is to do this. Aggregate Mask and Match bits 0 and 1 are available, so why not have them driven directly from such a register.

6. Clarify When CTB Has New Contents. Currently it's a little hassle to do sanity checks that your CTB really got new contents from running your LAC program. Especially when you are wondering if you configured everything correctly. You can "trust that a good-status completed LAC program means you have new CTB contents". You can alternate the CTB's external mux between what you want to collect and something else, then read-out the CTB and see that contents changed.

7. CTB Zeroing. An SCB-register "ClearCtb" action-bit in each CTB, that would zero-out the CTB (taking 1024 clocks). This bit could be readable and self-clears after the 1024 clocks have passed, so it's safe to start a new collection.

8. StopOnFull Final Address. Currently, in StopOnFull mode, when the CTB gets full and stops collecting, the final address is 0x000, which is the same address it would have if it never started! Either change this to stop at 0x3FF, or have a sticky overflow bit which clears when you write WtAddrClr in R_CtbxColCtl.

9. StoppedOnFull Status Bit. If in StopOnFull mode, have a read-only bit StoppedOnFull in R_CtbxColCtl. This signal already exists in the CTB Verilog code.

10. Fix the "Collecting" Status Bit. Bit "Collecting" of R_CtbxColCtl is directly flopped off of lac_ctb_CollectTrace_c0a, which means it doesn't take into consideration a CTB in StopOnFull mode that has become full. Reading of the CTB works in that case. Change Collecting to be false if StopOnFull and full. A signal with this info is available in the CTB verilog code. You might also consider having "Collecting" read back as 0 when EnableCollect==0. To be able to see the level of signal lac_ctb_CollectTrace_c0a clearly in one central place, add read-only bit "CollectTrace" to R_LacCtl (or if R_LacCtl gets broken-up into several registers as suggested, put this bit in whatever register contains the other read-only fields).

11. Have 0xFFFFFFFF Indicate Bad Read. If you try to read the contents of your CTB when you cannot, you currently get all-zeros. All-zeros can mean you never collected anything, and also for some units it's a likely read-result if you collected during an idle time. A tiny change in the verilog could make it return 0xFFFFFFFF's for reads when you can't read the CTB. This would be clearly different than a failure to trigger collection, and is an almost-impossible long series of values for any CTB to collect.

12. Stopping LAC Stops Collection. Have a transition of the GO bit 1 -> 0 cause the CLRCOLL action. This eliminates the hazard of someone stopping the LAC program manually by clearing the GO bit, but then being unable to read any CTB contents because CollectTrace is still ON. Have this be by 1 -> 0 transition, not by GO==0, so we can have the previously-mentioned registers that turn on and off collection. The way OLCA is now it can be very irritating if you happened to shut off LAC by writing 0 to the GO bit when collection was ON. There's no straightforward way to shut off collection of all enabled CTBs by register-write, you can only shut them off by opcode CLRCOLL in a running LAC program. This is no problem when the next LAC program you wish to run is of the CTB StopOnFull=0 unqualified style, but if you are doing qualified collection with StopOnFull=1 and you want to start at CTB address=0 it can be a problem. You might think you could just begin every LAC program with a CLRCOLL and your problems would be solved, but there's no way inside a LAC program to clear a CTB's WtAddr.

13. Move Delay Registers into the Trigger Blocks. Having the Delay Registers centralized in LAC means they're all flopped in cclk domain. FSW triggers and trigger blocks are in sclk domain. To be able to line-up FSW signals into a complex trigger is hard, although this was partly solved by providing some FSW trigger signals to it's trigger blocks more than once, with different sclk delays. The best solution to this is to have the delay registers in the Trigger Blocks, not centralized in the LAC.

14. More External-Mux Values, or Extra Mux in FSW. Boost the number of bits to control external muxes from 3 to 4 or 5. Do this for all types of trigger and collector blocks. Almost no extra logic is created by this except in those blocks where the extra external-mux-value options are used. The motivation for this is with regard

to the Link side of FSW. Currently OCLA in FSW only looks at FLR-0 and FLT-0 signals, due to mux-value limitations. For better board and system debug, to use OCLA freely to see damaged traffic arriving any one particular link, we really want all 6 links covered by OCLA. (b) Another way to get all 6 Link interfaces in FSW into OCLA, without changing OCLA Trigger or Collector blocks, is to just put a new register into FSW. This register in FSW's register address space would take values of 0, 1, or 2, and would drive a first level of muxing, selecting which link-number provides FLR and FLT signals to the current OCLA-register-driven external muxes.

15. More Collection Qualifiers. CTBs currently allow up-to 2 Qualifier signals. In some uses of CTBs there were more signals that would be handy to have available as qualifiers. The external mux selecting data for a CTB often selects between a good number of unrelated interfaces. In a number of cases you just accept that you have to do un-qualified collection, because the 2 qualifiers provided are not relevant to the interface or signals you are looking at.

16. More CTB Qualifier Inputs. Perhaps 4.

17. Use External Mux on Qualifiers. When instantiating CTBs, follow the example of how FSW Vector Trigger Blocks are instantiated, where the external mux selectors vary both the data ***and*** the qualifier to be used.

18. Eliminate Qualifiers in Codeword Trigger Blocks. The way Codeword Trigger Blocks work, all the trigger inputs are effectively qualifiers on each other. There's no reason to handle some inputs differently and call them "qualifiers".

19. Widen Vector Trigger Blocks to 64-Bits. FSW is really the only place where Vector Trigger Blocks are used, because the way they're used in DMA is more naturally served by Codeword Trigger Blocks. In FSW the natural width of the busses looked-at is 64 bits. It would be a usage simplification if the Trigger Block just looked at the 64 bits.

11.3 Description

In the ICE9 implementation, the OCLA units spread over the chip are:

- 1 LAC central controller.
- 6 CTBs – One for each of the six processor/L2 cache segments (PSXs).
- 2 CTBs – One for each of the two coherence engines (COHE and COHO).
- 2 CTBs – In the FSW unit.
- 1 CTB – In the PMI.
- 1 CTB – In the DMA Engine.
- 12 TRBCs – One for each of the six PSX segments, plus two for the PMI, plus one each for the COHE, COHO, DMA, and FSW.
- 3 TRBVs – Two in the FSW, and one in the DMA.

All CTBs are 1024 entries deep by 33 bits wide.

The number of different sets of signals you can choose to collect is quite large, selected by External Mux settings in each CTB:

- PSx CTBs: 3 mux settings * 6 PS's = 18 sets of signals.
- COHx CTBs: 4 mux settings * 2 COH's = 8 sets of signals (plus free-running counter).
- FSW Input CTB: 5 sets of signals.
- FSW Output CTB: 5 sets of signals.
- DMA CTB: 4 sets of signals.

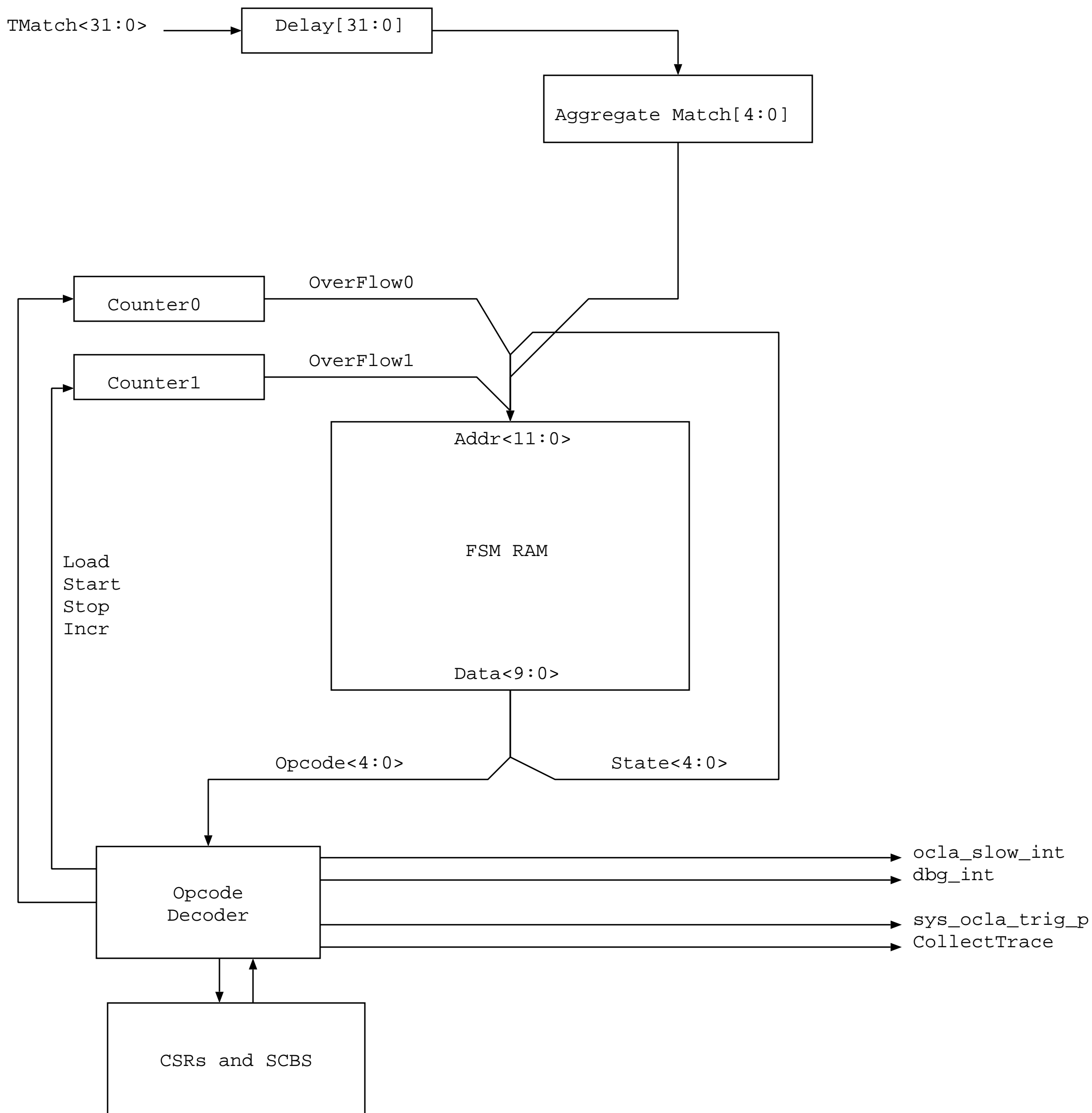


Figure 11.1: The On Chip Logic Analyzer Control Unit (LAC)

- PMI CTB: 7 sets of signals.

For a total of 47 sets of signals.

More than one CTB can be enabled for collection at once, although this only makes sense if you can arrange to have a window of time during which both CTBs are collecting meaningful events.

11.4 Package Attributes

Package

chip_ocla_spec

11.5 LAC Signals and Innards

11.5.1 What LAC Does

The main purpose of LAC (and your LAC program, and the values you write into LAC registers) is to control the CollectTrace signal (lac_ctb_CollectTrace_c2a) leading to all the Collector Blocks. When CollectTrace is ON, all CTBs (collector blocks) will collect values in the manner in which each has been configured. When CollectTrace is OFF, all CTBs are not collecting.

Secondary purposes of LAC are to set the Debug Interrupt, set the Slow Interrupt, set the 2 readable Flags, and to provide final status information to the user by ending at different addresses which can be read from the R_LacCtl register.

11.5.2 LAC Innards

The LAC provides the coordination and recognition of the actual trigger event. In most cases, logic analyzer triggers are more complicated than "fire when you see address X on bus Y." Instead, they frequently take the form of "fire on event A followed by event B followed by event C, but reset the recognizer on event D." This event recognizer is a state machine. I have no idea what sequences will be useful at this time, and I doubt any apriori guess is worthwhile. That being the case, the LAC is implemented as a field programmable state machine. (Don't worry, this isn't as complicated as it sounds.) The state machine may have up to 32 states.

The LAC has 32 trigger event inputs. Each input is synchronized and passes through a programmable delay chain that imposes between 2 and 7 cclk cycles of delay. The 32 bit vector that pops out of the array of delay chains is compared (using value/mask pairs similar to those in a vector comparison TRB – see section 11.8) in five aggregate event comparators. This a five bit wide "trigger event vector."

The LAC also contains two 16 bit counters (12 bits in Ice9A). Each counter is loaded with an initial value that is scanned in when the LAC is started. The initial value is written to counter X when the state machine selects the LOADx opcode or whenever the 'Go' bit in the LAC Control Register is set to one. When the counter overflows, it sets the CTRxOFLO bit. This bit is sticky; it stays set until either the recognizer asserts STARTx or LOADx again or the 'Go' bit in the LAC Control Register is set to one which forces a LOAD to both counters. Figure 11.1 shows the outline of the control unit. The FSM RAM holds 4K ten bit instructions. An instruction consists of both an opcode and a next state. The LAC is configurable to implement any state machine possible with seven inputs, five outputs and thirty two states.

Aggregate Match inputs are use to consolidate multiple trigger inputs to the LAC into a single pattern to be matched. See 11.5.3.3 on page 542. AMatch[x] is true if TMatch[31:0] & AMask[31:0] == AMatch[31:0].

11.5.2.1 LAC to SCB-Performance-Counters

All triggers coming into LAC from Trigger Blocks are also provided to the SCB Performance Counters mechanism as events to be counted.

To select a trigger from LAC to count in SCB Performance Counter, program the event field in R_ScbPerfBuckets, as described in the Serial Configuration Bus chapter. In "event" put the SubChipId for LAC (from the Addressing chapter) and 8 bits saying which one of the 32 triggers you want (bits 7,6,5 are zero).

These triggers are provided to SCB Performance Counters after being delayed by LAC's delay registers, but before being combined into t0 - t4. These delays allow SCB Performance Counters to condition one event on another event with a corrective skew between the two, in case the signals are related but occur one or more clocks apart. The conditioning is done by logic within the SCB Performance Counters mechanism. See the Serial Configuration Bus chapter for how to do this.

To provide these events, LAC hardware uses the performance counter feature of it's embedded SCB slave. The slave provides two input signals (x_scbs_event_x[1:0]), and a mux select for each (scbs_x_eventId{0|1}_xr). The LAC uses each mux select to choose one from among the 32 synchronized and delayed trigger inputs as specified above.

How much does this limit simultaneous normal use of OCLA? A little bit. One or two trigger blocks (and their delays) would be configured in the manner needed for SCB Performance Counters. An OCLA program could ignore them, using triggers from other blocks, but if it wants trigger from those blocks, they must use them with the same configuration and delays needed by SCB Performance Counters. Each Trigger Block has 2 trigger outputs, so if SCB Performance Counters only needs one of them, the other could be configured as needed for OCLA, although the external mux setting would have to be the same for both uses.

When counting events from trigger blocks in a different clock domain than LAC, like from FSW, it's better, when possible, to get the events directly from those trigger blocks. SCB Performance Counters has a way of getting correct counts from SCB slaves in different clock domains, whereas the clock-domain crossing from trigger blocks to LAC is not so nice. The PulseStretch mechanism for making sure LAC sees a trigger from a faster-clock trigger block is fine for triggers, but poor for counting. If you must get your counts from LAC, consider using OCLA PulseStretch along with the "transitions counting" option in SCB Performance Counters.

11.5.2.2 SCB-Performance-Counters to LAC

The 2 events configured to be counted by the SCB Performance Counters mechanism are also provided to OCLA for triggering. See the ScbTrig0En and ScbTrig1En fields in R_LacCtl. These are OR-ed into triggers t0 - t4, after trigger-block triggers are masked and matched, but before possible inversion by R_LacCtl field InvAgMat.

If you do this you'll have to manage your SCB Bus use. As explained in the Serial Configuration Bus chapter, anytime you are doing SCB writes or reads the detections of Performance Counter events are temporarily shut off. Even something as innocent as polling R_LacCtl.Flag to see whether OCLA got the trigger and collected will create blackout periods that could hide the very trigger you are waiting for!

How much does this limit simultaneous normal use of SCB Performance Counters? A lot. If you configure for 2 events, SCB Performance Counters would be limited to counting these events only. If you configure for one event from SCB Performance Counters to affect LAC programs, you still have some flexibility for unrelated use of SCB Performance Counters.

11.5.2.3 LAC Operation Codes

Enum

LacOp

Constant	Mnemonic	Definition	Product
5'h0	NOOP	Do Nothing in Particular	
5'h4	SETEXTP	Set External OCLA trigger output pin	
5'h5	CLREXTP	Clear External OCLA trigger output pin	
5'h6	SETCOLL	Set CollectTrace output	
5'h7	CLRCOLL	Clear CollectTrace output	
5'h8	SETFL0	Set Flag 0 in CSR	
5'h9	CLRFL0	Clear Flag 0 in CSR	
5'ha	SETFL1	Set Flag 1 in CSR	
5'hb	CLRFL1	Clear Flag 1 in CSR	
5'hc	SETDBI	Set Debug Interrupt output	
5'he	SETSLI	Set Slow Interrupt output	
5'h10	START0	Start Counter 0	
5'h11	STOP0	Stop Counter 0	
5'h12	LOAD0	Load Counter 0	
5'h13	INCR0	Increment Counter 0	
5'h14	START1	Start Counter 1	
5'h15	STOP1	Stop Counter 1	
5'h16	LOAD1	Load Counter 1	
5'h17	INCR1	Increment Counter 1	
5'h18	INCRBTH	Increment Both Counters	ICE9B+

11.5.2.4 Be Sure To Shut Off CollectTrace

If your LAC program will be or might be used on Ice9A chips, it needs to shut off CollectTrace before the program finishes or is stopped by register-write. Otherwise it may (a) cause you to get all-zeros when you read the contents of a collector block, of collector-block contents, or (b) cause premature data collection during the next use of OCLA. This is fixed in Ice9B and later, but in Ice9A, stopping the LAC program does not shut off CollectTrace. In Ice9A it can only be shut off by a LAC program instruction. See the "CTB Innards" section below for more details.

11.5.3 LAC Registers

11.5.3.1 The Control Register

Register

R_LacCtl

Attributes

-writeonemixed

Address

0xE_6800_0000

Bit	Mnemonic	Access	Reset	Type	Definition
31:27	ScbTrig1En	RW	0		OR scb_ocla_event_cr[1] into AgMatch[x]
26:22	ScbTrig0En	RW	0		OR scb_ocla_event_cr[0] into AgMatch[x]
21:17	InvAgMat	RW	0		Invert sense of AgMatch. When [x] is True, AgMatch[x] = ((TrigIn[31:0] & AMask[31:0]) != AMatch[31:0])
16	DbgInt	RW1C	0		Debug Interrupt to MIPS Cores
15	SlowInt	RW1C	0		Slow Interrupt output
14:3	FSMAddr	R	0		Current state of Address input to FSM RAM
2:1	Flag	R	0		Readable flags from the FSM Outputs
0	Go	RW	0		When TRUE, FSM is sequencing. When Go is 0, the STATE is set to 0 and the opcode is 0 (NOOP). When Go transitions to 1, the initial STATE is 0.

Be careful, when writing GO=1 to start the LAC program: That same register-write must contain your desired configuration values for ScbTrig1En, ScbTrig0En, and InvAgMat.

11.5.3.2 The Delay Registers

Each input trigger passes through two levels of CCLK flops (as a synchronizer). Each trigger then can be delayed by from 0 to 5 additional CCLK cycles before passing on to the AggregateMatch comparators. See "Uses for the Delay Registers" subsection of "Hints for Using Trigger Blocks" section later in this chapter. If you put a 6 or 7 in, you get a delay of only 5.

Register

R_LacTrgDly[31:0]

Address

0xE_6800_0100-0xE_6800_017f

Bit	Mnemonic	Access	Reset	Type	Definition
2:0	Dly	RW	0		Select Delay for trigger input [x]

11.5.3.3 The Aggregate Mask Registers

Register

R_LacAggMsk[4:0]

Address

0xE_6800_0600-0xE_6800_0613

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Mask	RW	0		Full mask register (Overlaps allowed)
31:30	TrbcPs5	RW	0		Processor Segment 5 Codeword Triggers (Overlaps allowed)
29:28	TrbcPs4	RW	0		Processor Segment 4 Codeword Triggers (Overlaps allowed)
27:26	TrbcPs3	RW	0		Processor Segment 3 Codeword Triggers (Overlaps allowed)
25:24	TrbcPs2	RW	0		Processor Segment 2 Codeword Triggers (Overlaps allowed)
23:22	TrbcPs1	RW	0		Processor Segment 1 Codeword Triggers (Overlaps allowed)
21:20	TrbcPs0	RW	0		Processor Segment 0 Codeword Triggers (Overlaps allowed)
19:18	TrbcCohe	RW	0		Even Coherence Unit Codeword Triggers (Overlaps allowed)
17:16	TrbcCoho	RW	0		Odd Coherence Unit Codeword Triggers (Overlaps allowed)
15:14	TrbvFswo	RW	0		Fabric Switch Output Vector Triggers (Overlaps allowed)
13:12	TrbvFswi	RW	0		Fabric Switch Input Vector Triggers (Overlaps allowed)
11:10	TrbcFsw	RW	0		Fabric Switch Control/Status Codeword Triggers (Overlaps allowed)
9:8	TrbvDma	RW	0		DMA MicroEngine Vector Triggers (Overlaps allowed)
7:6	TrbcDma	RW	0		DMA CSW Bus Stop Codeword Triggers (Overlaps allowed)
5:4	TrbcPmii	RW	0		PMI Internal Signal Codeword Triggers (Overlaps allowed)
3:2	TrbcPmi	RW	0		PMI CSW Bus Stop Codeword Triggers (Overlaps allowed)
1:0		RW	0		Reserved (Overlaps allowed)

11.5.3.4 The Aggregate Match Registers

Description

Match against incoming masked delayed triggers. Aggregate match X occurs with (DelayedTrigger[31:0] & Mask[X]) == Match[X]. Defaults to nonzero value so that the match always fails until configured.

Register

R_LacAggMat[4:0]

Address

0xE_6800_0640-0xE_6800_0653

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Match	RW	0xffffffff		Full match register (Overlaps allowed)
31:30	TrbcPs5	RW	0x3		Processor Segment 5 Codeword Triggers (Overlaps allowed)
29:28	TrbcPs4	RW	0x3		Processor Segment 4 Codeword Triggers (Overlaps allowed)
27:26	TrbcPs3	RW	0x3		Processor Segment 3 Codeword Triggers (Overlaps allowed)
25:24	TrbcPs2	RW	0x3		Processor Segment 2 Codeword Triggers (Overlaps allowed)
23:22	TrbcPs1	RW	0x3		Processor Segment 1 Codeword Triggers (Overlaps allowed)
21:20	TrbcPs0	RW	0x3		Processor Segment 0 Codeword Triggers (Overlaps allowed)
19:18	TrbcCohe	RW	0x3		Even Coherence Unit Codeword Triggers (Overlaps allowed)
17:16	TrbcCoho	RW	0x3		Odd Coherence Unit Codeword Triggers (Overlaps allowed)
15:14	TrbvFswo	RW	0x3		Fabric Switch Output Vector Triggers (Overlaps allowed)
13:12	TrbvFswi	RW	0x3		Fabric Switch Input Vector Triggers (Overlaps allowed)
11:10	TrbcFsw	RW	0x3		Fabric Switch Control/Status Codeword Triggers (Overlaps allowed)
9:8	TrbvDma	RW	0x3		DMA MicroEngine Vector Triggers (Overlaps allowed)
7:6	TrbcDma	RW	0x3		DMA CSW Bus Stop Codeword Triggers (Overlaps allowed)
5:4	TrbcPmii	RW	0x3		PMI Internal Signal Codeword Triggers (Overlaps allowed)
3:2	TrbcPmi	RW	0x3		PMI CSW Bus Stop Codeword Triggers (Overlaps allowed)
1:0		RW	0x3		Reserved (Overlaps allowed)

11.5.3.5 The Initial Counter Value Registers

Register

R_LacIniCtr[1:0]

Address

0xE_6800_0700-0xE_6800_0707

Bit	Mnemonic	Access	Reset	Product	Definition
15:0	InitValB	RW	0	ICE9B+	Value to be loaded into counter [x] when Reload[X] is true in ICE9B or later. (Overlaps allowed)
11:0	InitVal	RW	0	ICE9A	Value to be loaded into counter [x] when Reload[X] is true in ICE9A or ICE9A1. (Overlaps allowed)

Note: In ICE9A and ICE9A1, bits 15:12 don't exist, will ignore writes, and read-back 0.

11.5.3.6 The Current Counter Value Registers

Register

R_LacCtr[1:0]

Address

0xE_6800_0710-0xE_6800_0717

Bit	Mnemonic	Access	Reset	Product	Definition
31	OverflowB	R	0	ICE9B+	The "current" state of the counter's overflow bit in ICE9B or later. Sets when bits 15:0 roll over. Won't clear if they roll over again.
30:16					Reserved
15:0	CountB	R	0	ICE9B+	The "current" state of the counter in ICE9B or later. (Overlaps allowed)
12	Overflow	R	0	ICE9A	The "current" state of the counter's overflow bit in ICE9A or ICE9A1. Sets when bits 11:0 roll over. Won't clear if they roll over again. (Overlaps allowed)
11:0	Count	R	0	ICE9A	The "current" state of the counter in ICE9A or ICE9A1. (Overlaps allowed)

Note: The actual sizes of the counters match the above fields for the stated versions of ICE9.

11.5.3.7 The FSM RAM

Class

LacRamAddr

Bit	Mnemonic	Definition
11	OverFlow1	Counter 1 Overflow
10	OverFlow0	Counter 0 Overflow
9:5	FsmState	FSM Next State
4:0	AgMatch	Aggregate Match

Register

R_LacRam[4095:0]

Address

0xE_6800_4000-0xE_6800_7fff

Bit	Mnemonic	Access	Reset	Type	Definition
9:5	State	W	0		Next state for the FSM.
4:0	Opcode	W	0		LAC Opcode

11.5.4 LAC Signals

The LAC contains its own SCB slave unit. It runs in the CCLK domain. Table 11.2 shows the various LAC input and output signals.

Signal	Clock	I/O	Description
reset_e1cr_l	cclk	In	Active-low reset, which deasserts synchronous with cclk.
(16x) trbN_lac_Trig_x2a[1:0]	various	In	Trigger block asserts this signal when the trigger condition is met. This must be synchronized to CCLK domain by the LAC. The synchronized and delayed version of these signals are also connected to the event wires of the local SCB slave
lac_xxx_SlowInt_c2a	cclk	Out	Connected to the slow interrupt
lac_xxx_DbgInt_c2a	cclk	Out	Connected to the MIPS debug interrupt
lac_xxx_ExtTrig_c2a	cclk	Out	External trigger pin (sys_ocla_trig)
lac_ctb_CollectTrace_c2a	cclk	Out	The LAC produces a single active-high signal telling all capture blocks to record data to their ring buffers.
scb_ocla_event_cr[1:0]	cclk	In	Events from SCB master
xxx_lac_scbs_id[6:0]	cclk	In	SCB ID (tied to AddrSubId::OCLA in BBS)
chaini_ctb_dat_r[2:0]	cclk	In	Serial chain SCB input
ctb_chaino_dat_r[2:0]	cclk	Out	Serial chain SCB output

Table 11.2: LAC Signals

11.6 Collector Blocks (CTBs) in general

This section describes what's common to all Collector Blocks. The signals collected by each individual Collector Block are described in later sections.

Each CTB is a trace buffer that is as large 32 bits wide and 1K entries deep. The actual size is configured based on the space available near the block. (Only the array size changes, all control registers are wide enough to accomodate a 32x1K trace memory.) The trace buffer data inputs are connected to the data stream that we want to observe. The trace buffer write port runs off the same clock that sources the observed data stream. Figure 11.2 shows the outline of a CTB. Its primary inputs are the SampleDataIn[31:0] signal and the CollectTrace input that indicates the trace buffer should collect data. When the central controller (LAC) detects that the trigger event has been satisfied, it will assert or deassert CollectTrace at the appropriate time to all the CTBs on the chip. At the deassertion edge of CollectTrace, the WT Addr in each CTB will be frozen. The CollectTrace signal from the LAC is timed to the L2 cache clock – cclk. CTBs connected to other clock domains are responsible for synchronizing this input to their own domain.

Capture blocks (CTB) are instantiated in or near the unit whose data they will sample, and they are clocked by the same clock as the data to be sampled. In the description below, I will use "xclk" to represent the local clock domain.

11.6.1 CTB Innards

Each CTB contains its own SCB slave, since this keeps things reasonably simple, and the size of the SCB slave is small compared to even the minimal CTB configuration.

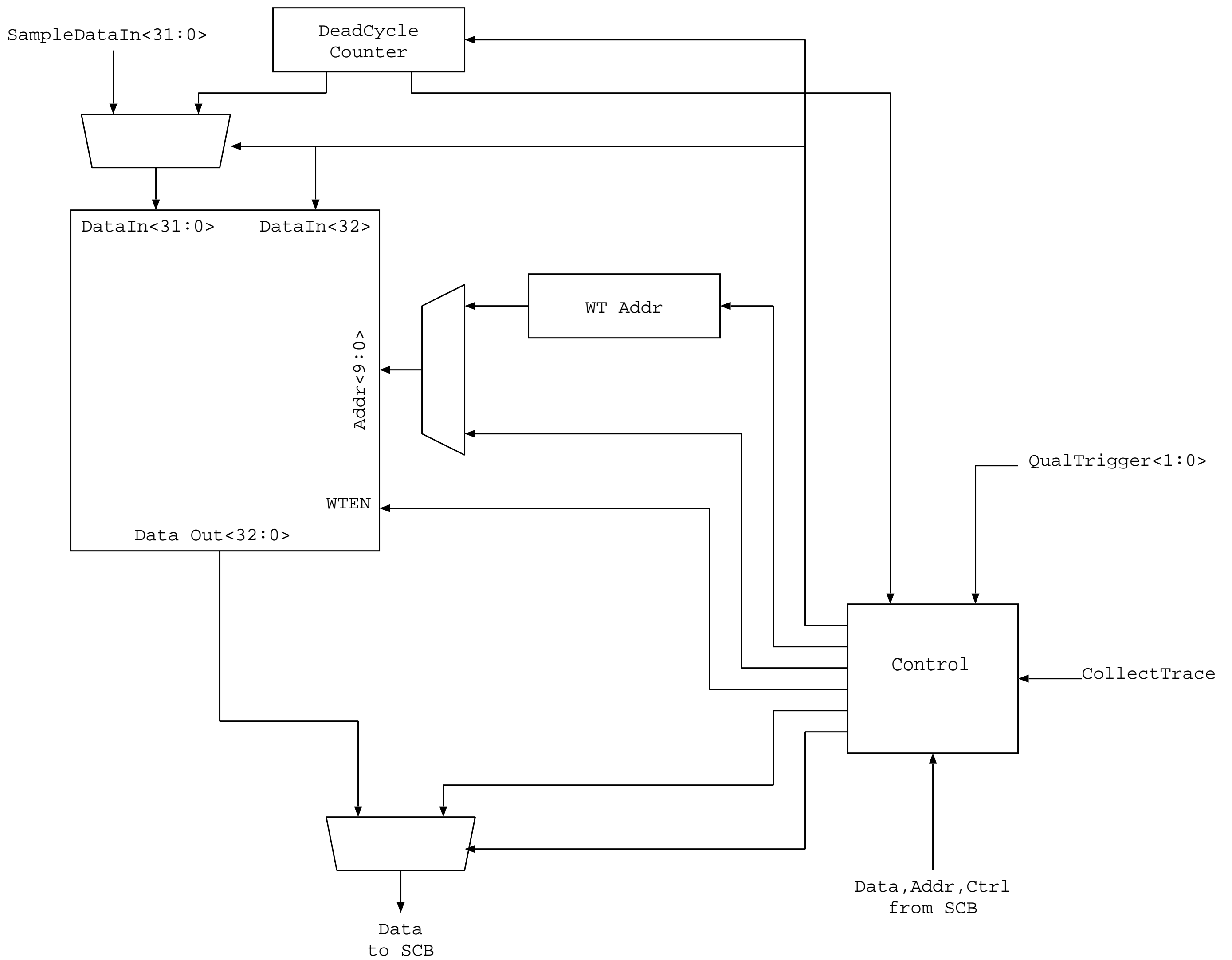


Figure 11.2: On Chip Logic Analyzer Capture Block (CTB)

11.6.1.1 The Control Unit and Muxes

The Control Unit contains the trace collection control register and is responsible for sequencing writes and reads from the trace RAM. It also recognizes dead collection cycles and manages the dead cycle counter.

The Output Mux selects between the low 32 bits of the trace RAM, the top bit plus the low 31 bits of the trace RAM, the WT Address register, or the contents of the collection control register. The choice is determined by the SCB register address.

11.6.1.2 The WT Addr Register

The WT Addr register can be cleared by the Control unit (see 11.6.2.1) and increments each time we write a sample or dead cycle count to the trace RAM.

11.6.1.3 The Dead Cycle Counter

Not all samples are worth collecting. All collector blocks except the one in PMI have a "qualified collection mode" (see 11.6.2.1).

When qualified collection is enabled, the trace will include counts of cycles in various locations instead of collector signals data. Trace entries that are cycle counts are marked by setting bit 32 in the trace RAM to 1. We can read bit 32 by reading the "topbits" register range. When the qualifying condition is not met, we are not collecting trace samples, instead we increment the DeadCycle counter on each such cycle and store it in the collector block memory without advancing the write address. Once a "qualified" clock occurs, write address is advanced and the normal collection data is stored. The dead cycle counter is cleared each time a new qualified sample is recorded into the trace RAM. This compacts or collapses what's stored in a collector block, allowing events over much more than the usual 1024 clocks to be observed.

The dead cycle counter is only 16 bits. Whenever it rolls-over, a 0xFFFF is stored, and the write address is advanced.

11.6.1.4 A Dead Cycle Counter Bug

(a) Dead Cycle counts are 1 too high. The smallest Dead Cycle count you'll see stored in a CTB is 2, which means 1 non-qualified clock. The largest you'll see is 0xFFFF, which means 0xFFFE non-qualified clocks.

(b) After rollover, after storing the 0xFFFF, the Dead Cycle counts stored are 1 too low. The smallest Dead Cycle count you'll see stored in a CTB is 0, which means 1 non-qualified clock. The largest you'll see is 0xFFFF, which means 0x10000 non-qualified clocks.

These corrections to what you read from a CTB apply to the usual LAC programs you are likely to write, where the LAC program has left collection turned on for a medium or long period of time, and the storing of dead cycle counts in the CTB is being controlled by the selected qualifier signal turning on and off. If you write a LAC program that turns on collection for a short period of time, and qualification is not met during that entire time, the stored dead cycle count will be correct. For example if you enabled collection for 5 clocks, and qualification was never met, you'd get a "5" stored.

11.6.1.5 The Trace RAM

The Trace RAM is configurable, and is at most 33 bits by 1K entries. In all cases, the width of the RAM is 1 bit wider than the input sample, to allow recording of "dead cycle" markers.

11.6.1.6 When Can You Read CTB Contents?

One of 3 conditions must be be true for you to read-out the CTB contents with SCB reads:

(1) Your LAC program has shut OFF CollectTrace. In Ice9A this can only be done by a LAC program instruction, no register write can do it, and stopping the LAC program does not do it. In Ice9B and later, stopping the LAC program will also shut OFF CollectTrace.

(2) The CTB in question is in StopOnFull mode, and has become full.

(3) You clear EnableCollect in the CTB's R_CtbxColCtl.

If none of these conditions have been met when you read-out the contents of a CTB, you will get all-zeros! This may give you the wrong idea that nothing was collected, or the wrong idea that you triggered and collected at a time when no activity was occuring on the signals being collected. To find out if CollectTrace is ON, read R_CtbxColCtl in any CTB, and look at bit "Collecting".

11.6.1.7 Do You Need To Shut-Off CollectTrace?

If you will-be or might-be running on Ice9A chips, and if your next use of OCLA has CTBs in StopOnFull mode, you probably want to shut-off CollectTrace (if it's on) before configuring and initializing for that OCLA run. If your next use of OCLA has CTBs is rollover mode (StopOnFull==0) then CollectTrace being ON doesn't matter.

Methods of shutting-OFF CollectTrace are described later in the OCLA Programming Suggestions section.

Why would CollectTrace be ON? In an Ice9A chip, the previously-run LAC program left it on, either due to a LAC program error, a trigger never occuring, or the LAC program was halted in the middle by a write of GO=0.

11.6.2 Registers

For "x" in the register names below, substitute desired collector name, from these:
Ps0, Ps1, Ps2, Ps3, Ps4, Ps5, Cohe, Coho, Fswi, Fswo, Dma, Pmi.

11.6.2.1 The Collection Control Register

Register

R_CtbxColCtl

Attributes

-writeonemixed

Address

0x00_0000 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
11:9	ExtMuxSel	RW	7		External Mux Select for logic outside the CTB to select alternate capture input sources. Many units use "7" to disable flops or data to their CTB. (see Note 2, Note 3, Note 4)
8	EnableCollect	RW	0		Collect Data when CollectTrace is asserted
7	Collecting	R	0		Will read as 1 when CollectTrace from LAC is asserted. Does not go to zero as you might expect when StopOnFull==1 and the CTB has become full. Also, it is unaffected by EnableCollect.
6	StopOnFull	RW	0		Stops collection when WtAddr overflows
5	DCtrClr	W1C	0		Clear Dead Cycle Counter – OBSOLETE, has no effect. (This definition kept for backward compatibility.)
4	WtAddrClr	W1C	0		Clear Write Address register. (see Note 5). Twc9 note: This bit should be moved to a different register, and -writeonemixed removed, as W1C mixed with normal write is annoying to SW.
3:2	QTrigState	RW	0		The values that QualTrigger1 and QualTrigger0 must be for collection, if qualification is enabled. You must leave these bits 0 if not enabling qualification.
1:0	QualTrig	RW	0		"Qualification Enable", with enables for QualTrigger1 and QualTrigger0

Note 1: In a given collector block, collection of values on signals from the unit occurs when 4 things are true: (a) R_CtbxColCtl.EnableCollect==1, (b) lac_ctb_CollectTrace_c0a==1 (the "Collect" signal from LAC), (c) R_CtbxColCtl.StopOnFull==0 or the collector block is not full yet, (d) "qualification" is currently satisfied. "Qualification" = ((QualTrigger-input-0 & QualTrig[0]) == QTrigState[0]) && ((QualTrigger-input-1 & QualTrig[1]) == QTrigState[1]).

Note 2: Actually, only COHe and COHo CTBs use the default value of 7 to disable activity (and see Note 3). Most other units just feed zeros in on the collection data inputs of their CTBs. Unusual cases: In PMI, all 8 ExtMuxSel settings, 0 - 7, are used for different sets of data to collect, except 5 which feeds zeros. In PSx, the lower-2 bits of ExtMuxSel choose between the 3 sets of data that can be collected, so ExtMuxSel settings 4 - 7 repeat the same choices of data as settings 0 - 3, with 3 and 7 collecting 0's for data. In Fswi and Fswo CTBs, mux settings 0 - 4 select different sets of signals, and mux settings 5, 6, 7 select the same data as muxSel=4.

Note 3: Due to a minor bug, in COHe or COHo, ***both*** the trigger block and collector block must have their muxes set to other than 7 to enable the external flops on signals coming into to ***either*** the trigger block or collector block.

Note 4: Due to Bug 1959, affecting PMI only in Ice9A, the ExtMuxSel field of R_TrbcPmiiTrigCtl must be used to select input signals for PMI's CTB, while the ExtMuxSel field in this register for PMI does nothing. This is fixed in Ice9B.

Note 5:

(a) Some usages of a CTB seem to get the CTB "stuck" when followed by other later uses of that CTB, which then fail to collect. This behavior is not fully characterized. We find that doing 2 writes to this register is best. Both writes have your new desired ExtMuxSel, QTrigState, QualTrig. The first write has WtAddrClr=1, EnableCollect=0. The 2nd write has EnableCollect=1 and your desired StopOnFull setting.

(b) You probably won't run into this, but: As described in BUG 2026, which is "Won't Fix" as of June 2006, any write to the SCB address-range of a specfic CTB, with bit-4 set in the write-data, will trigger R_CtbxColCtl.WtAddrClr, clearing that CTB's R_CtbxWtAddr. Although, since there are no other writable registers in a CTB, software should not be doing writes to any SCB address other than R_CtbxColCtl, within a CTB.

11.6.2.2 The RAM Lowbits

Register

R_CtbxRamLo[1023:0]

Address

0x00_1000-0x00_1fff (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	LoData	R	0		Low 32 Bits of Trace RAM (RAMData[31:0])

11.6.2.3 The RAM Highbits

Register

R_CtbxRamHi[1023:0]

Address

0x00_2000-0x00_2fff (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	HiData	R	0		Bits of Trace RAM including the dead-cycle-count marker (RAMData[32,30:0]). You don't get to see collected bit-31, but you do get to see the "dead cycle marker".

11.6.2.4 The Write Address

Register

R_CtbxWtAddr

Address

0x00_0010 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
9:0	WtAddr	R	0		Current (Next) Write Address. To clear this, write 1 to WtAddrClr bit in R_CtbxColCtl. For a CTB in StopOnFull mode, this will read back as 0 after the CTB has become full.

This is an index into the 1024-entry Collector Ram. After collecting for awhile and then stopping collecting, the last entry collected will be at WtAddr-1, or at index 0x3FF if WtAddr=0.

Can you tell from the value in this register whether collection occured? If the value is not zero, then some collection ***did*** occur since the last time that bit WtAddrClr in R_CtbxColCtl was written to 1. But if the value is zero, you can't tell. If StopOnFull=1 and enough is collected to fill the Collector Ram, then WtAddr will be back to zero again. If StopOnFull=0 and collection occurs for quite awhile then stops, you'll usually see a non-zero WtAddr, but there's 1 chance in 1024 that it will be zero.

11.6.3 CTB Signals

CTBs ("Collector Blocks" or "Capture Blocks") provide samples of the important signals within functional blocks of the ICE9 that would be difficult to observe in a running system. The CTBs reside logically within the functional blocks of the units they are sampling and are instantiated in or near the unit whose data they will sample, and they are clocked by the same clock as the data to be sampled. In the signal names below, I will use "xclk" to represent the local clock domain. Each of the CTBs is connected to its own SCB slave unit.

Signal	Clock	I/O	Description
reset_e1xr_l	xclk	In	Active-low reset, which deasserts synchronous with xclk.
xxx_ctb_SampleDataIn_x0a[31:0]	xclk	In	Data to be sampled
xxx_ctb_QualTrigger_x0a[1:0]	xclk	In	When the CTB is placed in Qualified Collection mode, these inputs control whether each sample is recorded or not. They should be tied high if this feature is not used.
lac_ctb_CollectTrace_c0a	cclk	In	The LAC produces a single active-high signal telling all capture blocks to record data to their ring buffers. The CTB must synchronize the signal to xclk before using it. All CTBs route CollectTrace through a dual-rank synchronizer.
ctb_xxx_SMuxSel_x1a[2:0]	xclk	Out	Selects from among alternate SampleData inputs. By convention, a mux select value of 7 indicates that the CTB is not in use, and that external flops related to the sample signals may have their clocks gated
xxx_ctb_scbs_id[6:0]	xclk	In	SCB Slave ID
chaini_ctb_dat_r[2:0]	xclk	In	Serial chain SCB input
ctb_chaino_dat_r[2:0]	xclk	Out	Serial chain SCB output

11.7 Hints for Using Collector Blocks

11.7.1 Collecting the Event You Triggered On

What you trigger-on is often what you want to collect and view. If you write your LAC program to branch on the trigger, then ***as fast as possible*** start collecting, you'll miss the event you want to see by many clocks! This is because the trigger signal takes several clocks to get through the trigger block, the LAC and your LAC program take several clocks to respond to a trigger and drive the collect signal, and then the collector block takes a couple clocks to start collecting.

The way to do this is:

1. Turn-on continuous collecting in the collector block, and enable collector-block address wrap-around.

2. Use the trigger in your LAC program to ***stop*** collecting, rather than to ***start*** collecting. If what you want to see is very short, just stop collecting when the trigger occurs.

3. If what you want to collect is longer than the delays involved with OCLA components, then either: [a] For a little extra time, put some extra steps in your LAC program between trigger and stopping collecting, or [b] For more extra time, when the trigger occurs start one of the timers, and when the timer overflows stop collection.

4. Find out where in the collector block the collection stopped by reading R_CtbxWtAddr (where "x" is your Ctb name). Then read a desired number of collector block entries leading up to (but not including) that collector block index, wrapping around from top to bottom of collector block, if needed.

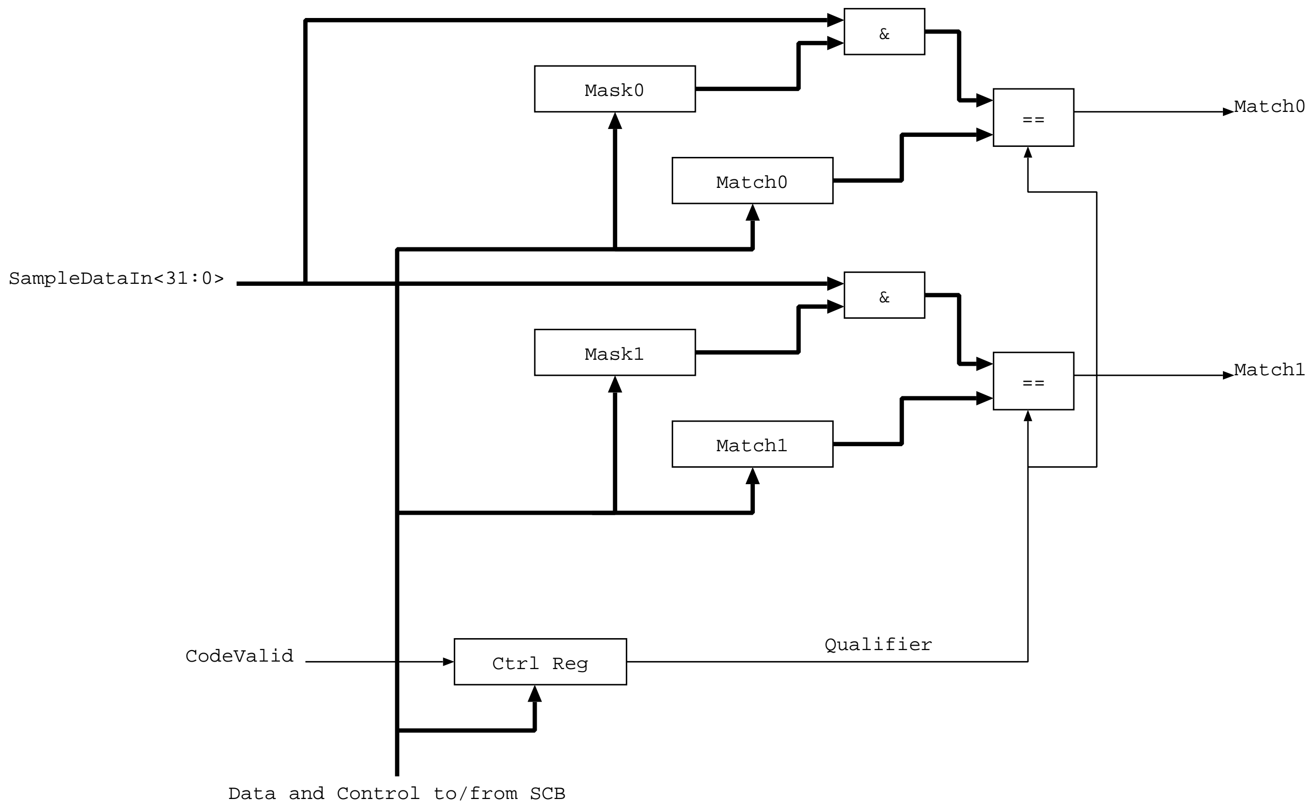


Figure 11.3: Vector Comparison Trigger Block

11.8 Vector Trigger Blocks (TRBVs) in general

This section describes what's common to all Vector Trigger Blocks. The signals available to trigger-on in each individual Vector Trigger Block are described in later sections.

Each TRBV provides a mechanism for trigger comparison between a 32 bit input vector and up to 32 bits of value and mask state to produce a TMatch signal. The TMatch output of the trigger block is synchronous with the clock domain of the input data. It is the responsibility of the LAC to resynchronize this signal into the cclk domain. The TMatch output is true when (INDat AND Mask) == Value. Since the TMatch output is synchronized to the source data clock, it may persist for too short a time to be sampled by the cclk in the LAC. In these cases, the TRB is responsible for ensuring that the TMatch/SMatch pulse width is sufficiently wide to be sampled by a cclk. For trigger blocks in clock domains that are faster than CCLK, the "PulseStretch" bit in the TrigCtl register should be set to guarantee that any trigger match pulse is at least two clock cycles long. PulseStretch can also make it easyer to get events from two different trigger blocks to coincide. Each TRBV has two match outputs. (See Figure 11.3.)

11.8.1 SCB Performance Counter Connections

In addition to providing triggers to the central OCLA LAC, each TRBV provides each of the 32 bits of SampleDataIn[31:0] to SCB Performance Counters as events to count. The SCB Performance Counters mechanism can focus on just 2 signals from SampleDataIn[31:0], or it can sweep across several selections of those 32 signals.

As described in the Serial Configuration Bus chapter, program the SubChipID (from the Addressing chapter) for the desired TRBV into bits 14:8 of a R_ScbPerfBuckets[255:0] "event" field, bits 7:5 must be zero, and bits 4:0 are bit-number in SampleDataIn[31:0].

What if you want to count how often some or all of SampleDataIn[31:0] matches a pattern? This can be done for OCLA triggering purposes by the TRBV, but the SCB Performance Counters hookup to a TRBV is limited to just 2 bits of SampleDataIn. You can count pattern matches by getting your events to count from LAC rather than directly from TRBV. LAC gives SCB Performance Counters the trigger outputs from all Trigger Blocks.

How much does this limit simultaneous use of a TRBV for OCLA? Very little. A separate pair of muxes is provided for this purpose, so all of the internals of the TRBV in question can be configured as needed for OCLA. Only the external mux must be the same for both purposes.

TRBV events sent to SCB Performance Counters are not stretched by R_TrbvxTrigCtl.PulseStretch. SCB Performance Counters has it's own way to get the correct number of counts even if it's in a different clock domain from the TRBV.

The hardware wiring of these signals to SCB Performance Counters is accomplished by feeding them into the SCB slave embedded in the TRBV.

11.8.2 Registers

For "x" in the register names below, substitute desired vector trigger block name, from these:
Fswi, Fswo, Dma.

11.8.2.1 The Trigger Control Register

Register

R_TrbvxTrigCtl

Address

0x00_0000 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
5	PulseStretch	RW	0		If set, all matches will be "repeated" in the xclk tic after the match was detected.
4:2	ExtMuxSel	RW	7		External Mux Select for logic outside the TRBV to select alternate trigger input sources. (see Note 1)
1	QTrigState	RW	0		If QualTrig, then this is the value that W1[0] must match
0	QualTrig	RW	0		Enable qualification of trigger by W1[0] for both trigger0 and trigger1

Note 1: Power conservation: The default mux select value of 7 indicates that the trigger block is not in use, and that external flops related to the sample signals may have their clocks gated. Of course, you'll be writing a value other than 7 in this field when you use any TRBV, because all instances of TRBVs have external muxes, and in no case does the value 7 select any input trigger sources.

11.8.2.2 The Trigger Mask Registers

Register

R_TrbvxTrigMask[1:0]
R_TrbvxTrigMask[0] controls Match0, R_TrbvxTrigMask[1] controls Match1.

Address

0x00_0010-0x00_0017 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Mask	RW	0		Selects which bits from SampleDataIn must match.

11.8.2.3 The Trigger Match Registers

Register

R_TrbvxTrigMatch[1:0]
R_TrbvxTrigMatch[0] controls Match0, R_TrbvxTrigMatch[1] controls Match1.

Address

0x00_0020-0x0027 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Match	RW	0xffffffff		The value that SampleDataIn must be, after masking by the above register, to cause the trigger. Defaults to nonzero value so that with a mask of zero, the match always fails until configured.

11.8.3 TRBV Signals

Trigger blocks (TRB) are instantiated in or near the unit whose data they will sample, and they are clocked by the same clock as the data to be sampled. In the signal names below, I will use “xclk” to represent the local clock domain. Each of the TRBs is connected to its own SCB slave unit.

Signal	Clock	I/O	Description
reset_e1xr_l	xclk	In	Active-low reset, which deasserts synchronous with xclk.
xxx_trbv_SampleDataIn_x0a[31:0]	xclk	In	Data to be sampled. These signals are also connected to the event wires of the local SCB slave, “W0[31:0]” for your selected Trigger Mux value, in the later sections on each vector trigger block.
xxx_trb_CodeValid_x0a	xclk	In	“Code valid flag” used as input to the Qualifier
trbv_lac_Match_x2a[1:0]	xclk	Out	The trigger block asserts each of these signals when the vector comparison against their respective mask/match registers is true and the Qualifier is satisfied. Asserted for two successive xclk tics if PulseStretch is set
trbv_xxx_SMuxSel_x1a[2:0]	xclk	Out	Selects from among alternate SampleData inputs. By convention, a mux select value of 7 indicates that the TRB is not in use, and that external flops related to the sample signals may have their clocks gated
xxx_trbv_scbs_id[6:0]	xclk	In	SCB Slave ID
chaini_scbs_dat_r[2:0]	xclk	In	Serial chain SCB input
scbs_chaino_dat_r[2:0]	xclk	Out	Serial chain SCB output

11.9 Codeword Trigger Blocks (TRBCs) in general

This section describes what’s common to all Codeword Trigger Blocks. The signals available to trigger-on in each individual Codeword Trigger Block are described in later sections.

Each Codeword TRB provides a mechanism for trigger comparison between up to four five bit codewords and up to three lists of “interesting” codes. For instance, the TRBC (shown in Figure 11.4) can be used to detect any READ operation directed at the COHE from a non-processor source by connecting CodeSample0 input to the COHE’s command input, and a CodeSample1 input to the TID input. (These connections are statically established.) We’d then load a 32 bit vector into Table0 with a 1 in each position corresponding to the code for a CSW Read operation. We’d load a Table1 with a vector selecting all TID codes that come from the DMA or PCI/BBS widgets. Assuming

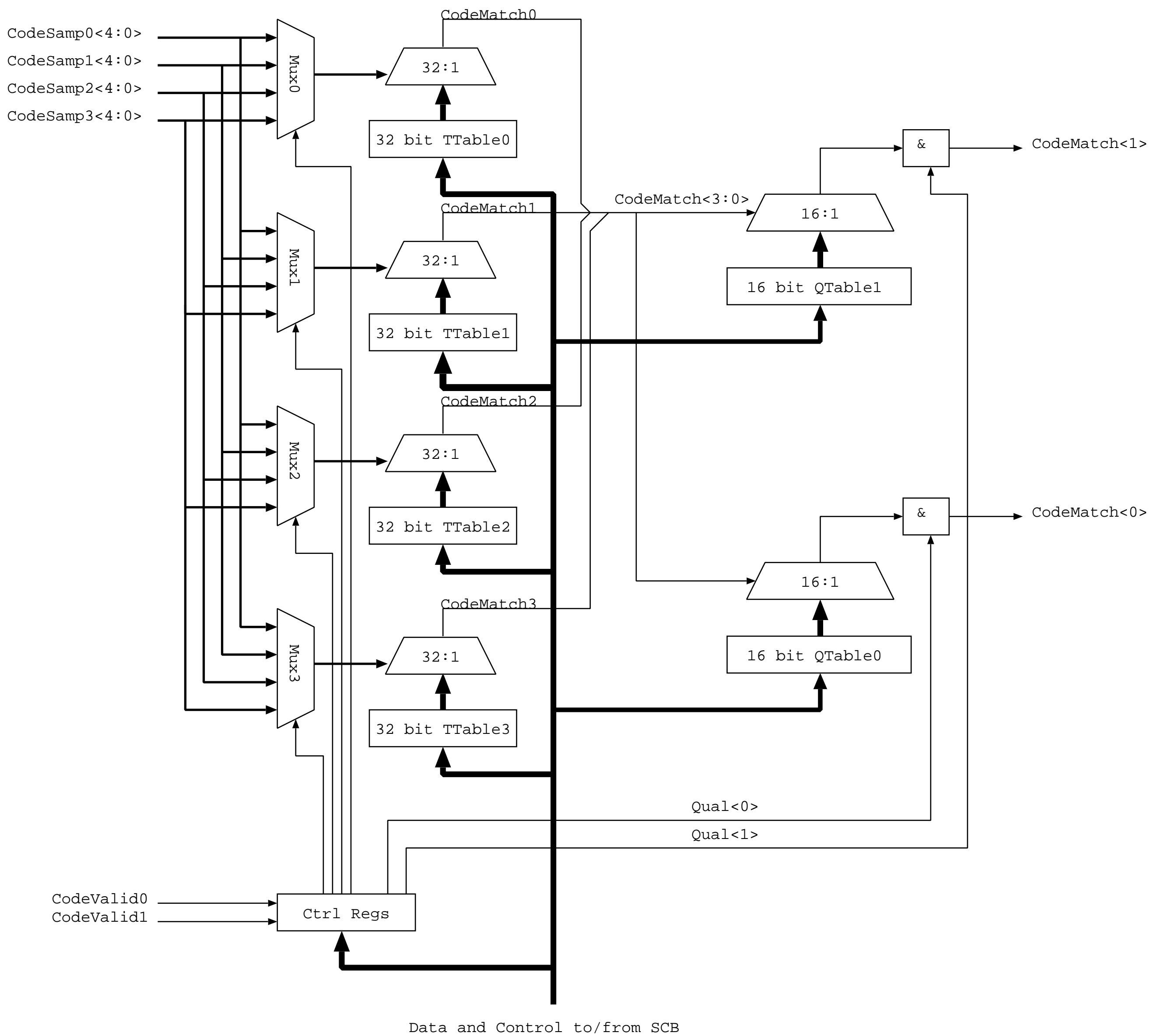


Figure 11.4: Codeword Trigger Block

the Qualifier condition is satisfied (see below) the CodeMatch output would be equal to 3 each time a read from a non-processor widget arrived at COHE.

CodeMatch may be qualified by looking at one or both of the “CodeValid” inputs. The control register selects which (or both) of the code Valid inputs are examined and what state they must be in to allow a match.

Note that any of the three tables can be “examined” by any of three of the four code inputs. This allows triggering on events such that the Code match word could be set up (for example) to produce 1 for READs, 2 for WRITEs, and 3 for RETRIEs.

The CodeMatch output of the trigger block is synchronous with the clock domain of the input data. It is the responsibility of the LAC to resynchronize this signal into the cclk domain. The TMatch output is true when (INDat AND Mask) == Value. Since the TMatch output is synchronized to the source data clock, it may persist for too short a time to be sampled by the cclk in the LAC. In these cases, the TRB is responsible for ensuring that the TMatch/SMatch pulse width is sufficiently wide to be sampled by a cclk. For trigger blocks in clock domains that are faster than CCLK, the “PulseStretch” bit in the TrigCtl register should be set to guarantee that any trigger match pulse is at least two clock cycles long. PulseStretch can also make it easyer to get events from two different trigger blocks to coincide.

Both bits of the CodeMatch output from the TRB are connected to the central LAC and to the x_scbs_event[1:0]

inputs of the associated SCB slave unit.

11.9.1 SCB Performance Counter Connections

Each TRBC provides its output triggers CodeMatch0 and CodeMatch1 to SCB Performance Counters as events that can be counted.

As described in the Serial Configuration Bus chapter, program the SubChipID for the desired TRBC (from the Addressing chapter) into bits 14:8 of a R_ScbPerfBuckets "event" field. Bits 7:0 of "event" are don't-cares.

TRBCs in a faster clock domain may need to use R_TrbcxTrigCtl.PulseStretch when sending triggers to LAC, but there's no need to PulseStretch when providing events to SCB Performance Counters. SCB Performance Counters will get the correct number of counts even if it's in a different clock domain from the TRBC. If you DO set PulseStretch, which you might want to if LAC needs the signals too, then SCB Performance Counters will get a much higher incorrect count. Note that there's only one PulseStretch bit, controlling both outputs.

How much does this limit simultaneous use of a TRBC for OCLA? If both CodeMatch0 and CodeMatch1 in a particular TRBC are used by SCB Performance Counters, then OCLA can only use that TRBC if it can use it with the exact same configurations. If only one CodeMatch is used by Performance Counters, then the other one can be configured as needed for OCLA, although the external mux and some of the internal muxes will have to be the same for both Performance Counters and OCLA. You can freely apply delays to these triggers within LAC, with no effect on them going to Performance Counters.

The hardware wiring of CodeMatch0 and CodeMatch1 to SCB Performance Counters is accomplished by wiring them to the embedded SCB slave within the TRBC. This is independent from the pathway by which LAC provides all of its trigger-block triggers to SCB Performance Counters.

11.9.2 Registers

For "x" in the register names below, substitute desired codword trigger block name, from these:
Ps0, Ps1, Ps2, Ps3, Ps4, Ps5, Cohe, Coho, Fsw, Dma, Pmi, Pmii.

11.9.2.1 The Trigger Control Register

Register

R_TrbcxTrigCtl

Address

0x00_0000 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
19	PulseStretch	RW	0		If set, all matches will be "repeated" in the xclk tic after the match was detected.
18:16	ExtMuxSel	RW	7		External Mux Select allows choice between multiple sets of trigger inputs feeding the same TRBC. (see Note 1) (see Note 2) (see Note 3)
15:14	Mux3Sel	RW	0		Mux 3 Input Select
13:12	Mux2Sel	RW	0		Mux 2 Input Select
11:10	Mux1Sel	RW	0		Mux 1 Input Select
9:8	Mux0Sel	RW	0		Mux 0 Input Select
7:6	QTMatch1	RW	0		Qual[1] = (CodeValid[1:0] & QT1Mask[1:0]) == QTMatch1[1:0]
5:4	QTMask1	RW	0		Enable Qualified Trigger mode for CodeValid 0 or 1 or both
3:2	QTMatch0	RW	0		Qual[0] = (CodeValid[1:0] & QT0Mask[1:0]) == QTMatch0[1:0]
1:0	QTMask0	RW	0		Enable Qualified Trigger mode for CodeValid 0 or 1 or both

Note: QTMatch1 and QTMask1 affect CodeMatch1, QTMatch0 and QTMask0 affect CodeMatch0.

Note 1: Power-saving: In most TRBC instantiations, where more than one set of trigger inputs is selected by ExtMuxSel, the default value of 7 indicates that the trigger block is not in use, and that external flops related to

the sample signals may have their clocks gated. Exceptions to this are the TRBCs in DMA and PMI which have only one set of input triggers, where the default value of 7 has no special meaning.

Note 2: Due to a minor bug, in COHe or COHo, ***both*** the trigger block and collector block must have their muxes set to other than 7 to enable the external flops on signals coming into to ***either*** the trigger block or collector block.

Note 3: Due to Bug 1959, affecting PMI only in Ice9A, the ExtMuxSel field of R_TrbcPmiiTrigCtl must be used to select input signals for PMI's CTB, while the ExtMuxSel field in R_CtbPmiColCtl does nothing. This is fixed in Ice9B.

11.9.2.2 The Trigger Table Registers

Register

R_TrbcxTrigTab[3:0]

Address

0x00_0010-0x001F (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	TTable	RW	0		Trigger Pattern for this Table

11.9.2.3 The Qualifier Table Registers

Register

R_TrbcxQualTab[1:0]

Address

0x00_0020-0x0027 (plus base address)

Bit	Mnemonic	Access	Reset	Type	Definition
15:0	QTable	RW	0		Trigger Pattern for this Table

11.9.3 TRBC Signals

Trigger blocks (TRB) are instantiated in or near the unit whose data they will sample, and they are clocked by the same clock as the data to be sampled. In the signal names below, "xclk" to represents the local clock domain. Each of the TRBs is connected to its own SCB slave unit.

Signal	Clock	I/O	Description
reset_e1xr_l	xclk	In	Active-low reset, which deasserts synchronous with xclk.
xxx_trbc_CodeSamp3_x0a[4:0]	xclk	In	Codeword3 to be tested. These signals are also connected to the event wires of the local SCB slave
xxx_trbc_CodeSamp2_x0a[4:0]	xclk	In	Codeword2 to be tested. These signals are also connected to the event wires of the local SCB slave
xxx_trbc_CodeSamp1_x0a[4:0]	xclk	In	Codeword1 to be tested. These signals are also connected to the event wires of the local SCB slave
xxx_trbc_CodeSamp0_x0a[4:0]	xclk	In	Codeword0 to be tested. These signals are also connected to the event wires of the local SCB slave
xxx_trbc_CodeValid1_x0a	xclk	In	One of two "code valid flags" used as input to the Qualifier
xxx_trbc_CodeValid0_x0a	xclk	In	One of two "code valid flags" used as input to the Qualifier
trbc_lac_CodeMatch_x2a[1:0]	xclk	Out	Each of these two bits is the selected bit from the corresponding QTable ANDed with the respective Qualifier bits. Asserted for two successive xclk tics if PulseStretch is set
trbc_xxx_SMuxSel_x1a[2:0]	xclk	Out	Selects from among alternate SampleData inputs. By convention, a mux select value of 7 indicates that the TRB is not in use, and that external flops related to the sample signals may have their clocks gated

Signal	Clock	I/O	Description
xxx_trbc_scbs_id[6:0]	xclk	In	SCB Slave ID
chaini_scbs_dat_r[2:0]	xclk	In	Serial chain SCB input
scbs_chaino_dat_r[2:0]	xclk	Out	Serial chain SCB output

11.10 Hints for Using Trigger Blocks

11.10.1 Using CodeValid Signals

The "CodeValid" or "Qualifier" signals hooked-up as inputs to most Vector and Codeword Trigger Blocks were conceived-of as a final "yes/no" on whatever other signals you've configured (by SCB) your Trigger Block to respond to. Unlike the other signals available for triggering, these are configured using bits in the main control register for your Trigger Block (the 2 Trig bits in R_TrbvxTrigCtl, or the 8 QT bits in R_TrbcxTrigCtl). But in use they're not really all that different from the other trigger inputs. Any Trigger Block input signal can effectively say "yes/no" on the overall trigger output from that Block. In a Collector Block, qualifiers play a very special role, but in a Trigger Block they're just one more signal which you can AND-into to the expression for one or both of the trigger outputs. They're just programmed differently.

11.10.2 Trigger Clock Domains

Almost all of OCLA operates in cclk, including LAC and most Trigger and Collector Blocks. Only the exception is that FSW Trigger and Collector Blocks are in sclk domain. sclk will always be slower-than or same frequency as cclk. No phase relationship is gauranteed between sclk and cclk, even when at the same frequency. Furthermore, when at the same frequency, there's a very small probability that on a signal going from sclk to cclk, a one-sclk-long pulse may not be seen at all in the cclk domain, due to over-time variations of on-which-cclk-edge the clock-syncronization logic decides to present a newly-changing sclk-domain signal. If Ice9 is operating with cclk faster than sclk this never happens, but you see occasional stretching of 1-sclk pulses from sclk domain becoming 2-cclks long in cclk domain.

Since triggers from Trigger Blocks are often 1 clock long, the loss of such a trigger pulse going from an FSW Trigger Block to the LAC would be a problem. The PulseStretch feature of Trigger Blocks provides a solution, making the trigger pulse 2 sclks long, which is sure to become at least one cclk long at the LAC.

FSW Trigger Blocks being in a different clock domain from LAC causes another problem. The delay regs in LAC cannot be used for FSW triggers as easilly or reliably as they can for the other Trigger Blocks.

11.10.3 Uses for the Delay Registers

The LAC has separate delay registers for each trigger signal coming from each Trigger Block. Here are some uses for them:

11.10.3.1 Aligning Mis-Aligned Signals From Same Trigger Block

Often you want to trigger on a combination of signals from a trigger block, that while related to the same one event, happen on different clocks, like when one of the signals asserts 1 or 2 clocks later than the others. Use the 2 trigger lines from that trigger block, one for each signal, then delay one of them in LAC. Either or both trigger lines could be from and-ed groups of signals.

11.10.3.2 Aligning CodeValid or Qualifier with Other Triggers in a Trigger Block

Line-up a signal or group of signals from a trigger block with the qualifier of that trigger block, if they differ by 1 or more clocks. Use one trigger line for the group of signals, unqualified. Use the other trigger line for the qualifier, qualifying "true" (mask=0, match=0).

11.10.3.3 Aligning Triggers from Different Trigger Blocks

This can compensate for one Trigger Block having more flops than the other between trigger signal source and LAC. This can adjust for an event in one Trigger Block occuring earlier than the related event in the other Trigger Block.

If the difference in time between these two triggers is too large for LAC's Delay Registers, you might be able to your LAC program to wait for the first event, then wait for the 2nd, with a timeout at which point it goes back to waiting for the first event. Of course this only works if "first events" are separated by enough clocks.

11.10.3.4 Provide Bigger Window for Coinciding Events

Combined with PulseStretch, provide a wider window of "coinciding" between single-clock events from different trigger blocks, up to 7 cclks wide! To do this use enable PulseStretch on both trigger blocks, and then send the same trigger out both trigger ports of each trigger block. In the delay registers, skew the 2 triggers from a given trigger block by 2 cclks relative to each other, providing a "trigger == true" time of 4 cclks from each trigger block. Use 4 Aggregate Matches to "and" each trigger from one trigger block with each trigger from the other trigger block. Then, in your LAC program, loop waiting to branch on any of these 4 Aggregate Matches to the same one "got the event" LAC state.

11.11 OCLA in use – PSx (Processor Segments)

The 6 Processor Segments have 1 Trigger Block each, and 1 Collector Block each. For "x" in "PSx" substitute each of 0,1,2,3,4,5.

11.11.0.5 Location of OCLA-PSx Blocks and Signals

PSx signals for OCLA triggering and collection are in the CAC part of each PSx.

From a usage point of view you don't need to know where the Trigger and Collector Blocks of OCLA-PSx are located, but if you are looking at the Verilog code, you might get confused, so here's the info: The Trigger Block for each PSx is located in it's CAC, but the Collector Block is located in one of the COH units. COHe contains 3 of the PSx Collector Blocks, and COHo contains the other 3. These 3 are not to be confused with COH's own Collector Blocks, which are connected to COH signals. Each of COHe and COHo contains one COH collector block and 3 PSx collector blocks.

11.11.1 PSx Triggers

Each of the Processor Segments will have a codeword trigger capable of detecting events coming from the CSW, and internal L2 controller state. We want to watch lots more signals than we have inputs for a TRBC, so we provide an external mux to select from between trigger sources that are hopefully not both interesting at the same time. The following tables define the codeword triggers for the most interesting signals and signal combinations in the ICE9 Cache. For the cache unit there are four mux selectable groupings of codeword triggers. Each class below represents one of the four mux selectable groupings. Note that all signals listed are flopped once before entering trigger blocks.

11.11.1.1 Processor Segment Trigger Mux 0

Class

TrbcPsxMux0

Attributes

-ocla -trbc -trbcpsx

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4:0]	LatCmd	xxx_trbc_CodeSamp0Mux0[4:0]	cac.lat_xxx_Command_c2a[4:0]	Command code for incoming request from C
W1[4:0]	LatCmdAddrTid	xxx_trbc_CodeSamp1Mux0[4:0]	cac.lat_xxx_CmdAddrTID_c2a[4:0]	Transaction ID for incoming request from C
W2[4:0]	LatDataTid	xxx_trbc_CodeSamp2Mux0[4:0]	cac.lat_xxx_DataTID_c4a[4:0]	The TID for the accompanying data from C
W3[4]	PsxToCswECmdAddrReq	xxx_trbc_CodeSamp3Mux0[4]	cac.psx_csw_ECmdAddrReq_c0a	Bid for evenbound bus to CSW
W3[3]	PsxToCswOCmdAddrReq	xxx_trbc_CodeSamp3Mux0[3]	cac.psx_csw_OCmdAddrReq_c0a	Bid for oddbound bus to CSW

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W3[2]	CswToPsxCmdAddrGnt	xxx_trbc_CodeSamp3Mux0[2]	cac.csw_psx_CmdAddrGnt_c1a	We got the last command cycle. cclk after psx_csw_ECmdAddrRe psx_csw_OCmdAddrReq_c0a
W3[1:0]		xxx_trbc_CodeSamp3Mux0[1:0]		Reserved always zero
W4[0]	Cv0LatCmdAddrValid	xxx_trbc_CodeValid0Mux0_x0a	cac.lat_xxx_CmdAddrValid_c2a	CSW is sending a command to PSX same cclk as lat_xxx_Command_c2a lat_xxx_CmdAddrTID_c2a
W5[0]	Cv1LatDataValid	xxx_trbc_CodeValid1Mux0_x0a	cac.lat_xxx_DataValid_c4a	Incoming Data-Valid from CSW

11.11.1.2 Processor Segment Trigger Mux 1

Class

TrbcPsxMux1

Attributes

-ocla -trbc -trbcpsx

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4]	SlcToTagBiuWrite	xxx_trbc_CodeSamp0Mux1[4]	cac.slc_tag_BiuWrite_cya	CPU to CAC request of a write, mem or IO
W0[3]	SlcToTagBiuRead	xxx_trbc_CodeSamp0Mux1[3]	cac.slc_tag_BiuRead_cya	CPU to CAC request of a read, mem or IO
W0[2]	SlcToTagIFetch	xxx_trbc_CodeSamp0Mux1[2]	cac.slc_tag_IFetch_cya	Instruction stream Fetch
W0[1]	CtlToSlcWinPrb	xxx_trbc_CodeSamp0Mux1[1]	cac.ctl_slc_WinPrb_c6a	This is a probe to L1 in response to a PRB-WIN from CSW or a victim displacement
W0[0]	CtlToSlcInvPrb	xxx_trbc_CodeSamp0Mux1[0]	cac.ctl_slc_InvPrb_c6a	This is a probe to L1 in response to a PRBINV from CSW. Ignore returned data.
W1[4:0]	LatCmdAddrTid	xxx_trbc_CodeSamp1Mux1[4:0]	cac.lat_xxx_CmdAddrTID_c2a[4:0]	Transaction ID for incoming request from CSW
W2[4:0]	CtlToLamPrbQState	xxx_trbc_CodeSamp2Mux1[4:0]	cac.ctl_lam_PrbQState_c4a[4:0]	Probe-queue handler state
W3[4:3]	SlcPrbDirty	xxx_trbc_CodeSamp3Mux1[4:3]	cac.slc_xxx_PrbDirty_cya[1:0]	Which of two 32 byte blocks in a probe were newly updated
W3[2:1]	SlcPrbDone	xxx_trbc_CodeSamp3Mux1[2:1]	cac.slc_xxx_PrbDone_cya[1:0]	Probe for both blocks has completed
W3[0]	SlcToCtlWbInProg	xxx_trbc_CodeSamp3Mux1[0]	cac.slc_ctl_WbInProg_czb	Writeback in progress
W4[0]	Cv0LatCmdAddrValid	xxx_trbc_CodeValid0Mux1_x0a	cac.lat_xxx_CmdAddrValid_c2a	CSW is sending a command to PSX
W5[0]	Cv1SlcToTagBiuMemAcc	xxx_trbc_CodeValid1Mux1_x0a	cac.slc_tag_BiuMemAcc_cya	CPU to CAC request address is a memory access

11.11.1.3 Processor Segment Trigger Mux 2

Class

TrbcPsxMux2

Attributes

-ocla -trbc -trbcpsx

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4:0]	LatCmd	xxx_trbc_CodeSamp0Mux2[4:0]	cac.lat_xxx_Command_c2a[4:0]	Command code for incoming request from CSW
W1[4:0]	LatCmdAddrTid	xxx_trbc_CodeSamp1Mux2[4:0]	cac.lat_xxx_CmdAddrTID_c2a[4:0]	Transaction ID for incoming request from CSW
W2[4]	SlcToTagBiuWrite	xxx_trbc_CodeSamp2Mux2[4]	cac.slc_tag_BiuWrite_cya	CPU to CAC request of a write, mem or IO
W2[3]	SlcToDatPrbWbVal	xxx_trbc_CodeSamp2Mux2[3]	cac.slc_dat_PrbWbVal_cya	Data in cz is a writeback from a probe

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W2[2]	SlcBiuPaused	xxx_trbc_CodeSamp2Mux2[2]	cac.slc_xxx_BiuPaused_c2b	Says SLC won't send new requests until pause deasserts
W2[1:0]	SlcPrbDone	xxx_trbc_CodeSamp2Mux2[1:0]	cac.slc_xxx_PrbDone_cya[1:0]	Probe for both blocks has completed
W3[4]	PrbRdReq	xxx_trbc_CodeSamp3Mux2[4]	cac.ctl_dat_PrbRdReq_c5a	Read a block out of the L2 and write it to the CSW
W3[3]	WtPrb2L2PrbState2	xxx_trbc_CodeSamp3Mux2[3]	[See_Note_1]	ctl_dat_WtPrb2L2_c5a ORed w ctl_dat_PrbState_c5a[2]
W3[2:1]	PrbState10	xxx_trbc_CodeSamp3Mux2[2:1]	cac.ctl_dat_PrbState_c5a[1:0]	Low 2 bits of PrbState (See Note 2)
W3[0]	LatDataValid	xxx_trbc_CodeSamp3Mux2[0]	cac.lat_xxx_DataValid_c4a	Incoming Data-Valid from CSW
W4[0]	Cv0LatCmdAddrValid	xxx_trbc_CodeValid0Mux2_x0a	cac.lat_xxx_CmdAddrValid_c2a	CSW is sending a command to PSX
W5[0]	Cv1SlcToTagBiuMemAcc	xxx_trbc_CodeValid1Mux2_x0a	cac.slc_tag_BiuMemAcc_cya	CPU to CAC request address is a memory access

Notes:

1. cac.ctl_dat_WtPrb2L2_c5a \|\| cac.ctl_dat_PrbState_c5a[2] in Ice9A. This was a mistake, bug 1995, which makes it hard to trigger on all 3 bits of ctl_dat_PrbState_c5a[2:0]. With only the lower 2 bits we can distinguish between four Cac State possibilities: 0=INV, 1=EXCL, 2=SHARE-or-DIRTY, 3=UPDATED. Signal ctl_dat_WtPrb2L2_c5a means for BRD writebacks to CSW, also write data to L2. Fixed in Ice9B to be just be cac.ctl_dat_PrbState_c5a[2], allowing triggering on all Cac States.
2. When the probe data is sent along, ctl_dat_PrbState_c5a is the state that should be propagated (all 3 bits, that is). PrbState is of type CacState, not CacPrbQState.

11.11.1.4 Processor Segment Trigger Mux 3

Class

TrbcPsxMux3

Attributes

-ocla -trbc -trbcpsx

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4:0]	PsxToCswCmd	xxx_trbc_CodeSamp0Mux3[4:0]	cac.psx_csw_Command_c0a[4:0]	Processor Segment to CSW Command
W1[4]	CtlToTagInvReq	xxx_trbc_CodeSamp1Mux3[4]	cac.ctl_tag_InvReq_c5a	Invalidate Request, reqAddr block should be invalidated
W1[3]	CtlToTagWinReq	xxx_trbc_CodeSamp1Mux3[3]	cac.ctl_tag_WinReq_c5a	In Biu pause, doing writeback & invalidate for PRBWIN from CSW
W1[2]	CtlToTagBrdReq	xxx_trbc_CodeSamp1Mux3[2]	cac.ctl_tag_BrdReq_c5a	In Biu pause, doing block-read for PRB-BRD from CSW
W1[1]	CtlToTagBwtReq	xxx_trbc_CodeSamp1Mux3[1]	cac.ctl_tag_BwtReq_c5a	In Biu pause, doing block-write for PRBBWT from CSW
W1[0]	CtlToTagShrReq	xxx_trbc_CodeSamp1Mux3[0]	cac.ctl_tag_ShrReq_c5a	In Biu pause, going to shared state for PRBSHR from CSW
W2[4]	SlcToTagBiuWrite	xxx_trbc_CodeSamp2Mux3[4]	cac.slc_tag_BiuWrite_cya	CPU to CAC request of a write, mem or IO
W2[3]	SlcToTagBiuRead	xxx_trbc_CodeSamp2Mux3[3]	cac.slc_tag_BiuRead_cya	CPU to CAC request of a read, mem or IO
W2[2]	SlcToDatPrbWbVal	xxx_trbc_CodeSamp2Mux3[2]	cac.slc_dat_PrbWbVal_cya	Data in cz is a writeback from a probe
W2[1]	SlcToTagBiuMemAcc	xxx_trbc_CodeSamp2Mux3[1]	cac.slc_tag_BiuMemAcc_cya	CPU to CAC request address is a memory access
W2[0]	SlcToTagIFetch	xxx_trbc_CodeSamp2Mux3[0]	cac.slc_tag_IFetch_cya	Instruction stream Fetch
W3[4]	TagToCtlW0Miss	xxx_trbc_CodeSamp3Mux3[4]	cac.tag_ctl_W0Miss_cza	(Tag-Miss on Way-0, or Idle) and not IO-access [See Note 2]
W3[3]	TagToCtlW1Miss	xxx_trbc_CodeSamp3Mux3[3]	cac.tag_ctl_W1Miss_cza	(Tag-Miss on Way-1, or Idle) and not IO-access [See Note 2]
W3[2]	TagToCtlPrbHit	xxx_trbc_CodeSamp3Mux3[2]	cac.tag_ctl_PrbHit_c6a	The incoming probe op hit on the L2

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W3[1:0]	TagToCtlBlkState	xxx_trbc_CodeSamp3Mux3[1:0]	cac.tag_ctl_BlkState_cza[1:0]	State of block we got a hit on
W4[0]	Cv0SlcToTagBiuMemAcc	xxx_trbc_CodeValid0Mux3_x0a	cac.slc_tag_BiuMemAcc_cya	CPU to CAC request address is a memory access
W5[0]	Cv1PsxToCswXCmdAddrReq	xxx_trbc_CodeValid1Mux3_x0a	[See_Note_1]	PSX to CSW Even or Odd Cmd Address Request

Notes:

1. cac.psx_csw_ECmdAddrReq_c0a || cac.psx_csw_OCmdAddrReq_c0a; // Request by Cac for either the evenbound or oddbound CSW Cmd Address Bus.
2. Bug2243: In Ice9A each of these "W0Miss, W1Miss" signals will be asserted when their "way" (W0 or W1) has a tag-miss on a Biu Memory Access, or anytime accessing tags is idle. This means they're similar to "~Hit" signals, except that for processor IO accesses, both of these will be 0 (which does not mean "Hit"). This is because tags are bypassed during IO accesses. To eliminate both Idles and IO-accesses, configure OCLA so that slc_tag_BiuMemAcc_cya must be true when looking for W0Miss or W1Miss to be either true or false. These trigger bits are improved in Ice9B to be tag_ctl_W0Hit and tag_ctl_W1Hit.
3. Bug2243: In Ice9A signals tag_ctl_W0Miss_cza and tag_ctl_W1Miss_cza are 1 cclock later than the other related signals provided, for a given access event. This means that to condition W0Miss or W1Miss with another signal you'll have to use both codeword trigger outputs, and then in LAC delay one relative to the other. This is fixed in Ice9B.

11.11.2 PSx Collectors

Each of the six PS CTBs contain the following mux inputs and signals.

11.11.2.1 PSx Input Collectors Qualifying Triggers

Class

CtbPsxQtrig

Attributes

-ocla -ctb -ctbpsx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
1	LatCmdAddrValid	xxx_ctb_QualTrig1_x0a	cac.lat_xxx_CmdAddrValid_c2a	CSW is sending a command to PSX
0	SlcToTagOp	xxx_ctb_QualTrig0_x0a	[See_Note_1]	CPU to CAC request of a read or write, mem or IO

Notes:

1. cac.slc_tag_BiuRead_cya || cac.slc_tag_BiuWrite_cya;

11.11.2.2 PSx Input Collector Mux 0

Class

CtbPsxMux0

Attributes

-ocla -ctb -ctbcac

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31	TagToCtlW1Miss	xxx_ctb_SampleDataIn0_x0a[31]	cac.tag_ctl_W1Miss_cza	(Tag-Miss on Way-1, or Idle) and not IO-access

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
30	TagToCtlW0Miss	xxx_ctb_SampleDataIn0_x0a[30]	cac.tag_ctl_W0Miss_cza	(Tag-Miss on Way-0, or Idle) and not IO-access
29:25	CtlToLamPrbQState	xxx_ctb_SampleDataIn0_x0a[29:25]	cac.ctl_lam_PrbQState_c4a[4:0]	Probe-queue handler state
24:21	SlcToLamRdyState1	xxx_ctb_SampleDataIn0_x0a[24:21]	cac.slc_lam_RdyState1_c2a[3:0]	Ready state from the SLC, pclk number 1 [See Note 1]
20	CswToPsxDataGnt	xxx_ctb_SampleDataIn0_x0a[20]	cac.csw_psx_DataGnt_c3a	Cache switch to processor segment data grant
19	PsxToCswODataReq	xxx_ctb_SampleDataIn0_x0a[19]	cac.psx_csw_ODataReq_c2a	Processor segment to cache switch odd data request
18	PsxToCswEDataReq	xxx_ctb_SampleDataIn0_x0a[18]	cac.psx_csw_EDataReq_c2a	Processor segment to cache switch even data request
17	CswToPsxCmdAddrGnt	xxx_ctb_SampleDataIn0_x0a[17]	cac.csw_psx_CmdAddrGnt_c1a	Cache switch to processor segment command grant
16	PsxToCswOCmdAddrReq	xxx_ctb_SampleDataIn0_x0a[16]	cac.psx_csw_OCmdAddrReq_c0a	Processor segment to cache switch odd command request
15	PsxToCswECmdAddrReq	xxx_ctb_SampleDataIn0_x0a[15]	cac.psx_csw_ECmdAddrReq_c0a	Processor segment to cache switch even command request
14	Always0	xxx_ctb_SampleDataIn0_x0a[14]	[Always_Zero]	Reserved
13:10	SlcToLamRdyState0	xxx_ctb_SampleDataIn0_x0a[13:10]	cac.slc_lam_RdyState0_c2a[3:0]	Ready state from the SLC, pclk number 0 [See Note 1]
9:5	LatCmdAddrTid	xxx_ctb_SampleDataIn0_x0a[9:5]	cac.lat_xxx_CmdAddrTID_c2a[4:0]	Command TID
4:0	LatCmd	xxx_ctb_SampleDataIn0_x0a[4:0]	cac.lat_xxx_Command_c2a[4:0]	Command

Notes:

1. The CPU runs on pclk, twice as fast as cclk, so for OCLA (in cclk) to see the sequence of ready states in the CPU, 2 successive pclk states are passed into Cac and into this collector block on each cclk. See RdyState1 in collector bits 24:21, and RdyState0 in collector bits 13:10. RdyState0 occurred in the CPU before RdyState1.

11.11.2.3 PSx Input Collector Mux 1

Class

CtbPsxMux1

Attributes

-ocla -ctb -ctbpsx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:22	LatAddrHi	xxx_ctb_SampleDataIn1_x0a[31:22]	cac.lat_xxx_Addr_c2a[35:26]	10 upper Address bits [35:26]
21:10	LatAddrLo	xxx_ctb_SampleDataIn1_x0a[21:10]	cac.lat_xxx_Addr_c2a[14:3]	12 lower Address bits [14:3]
9:5	LatCmdAddrTid	xxx_ctb_SampleDataIn1_x0a[9:5]	cac.lat_xxx_CmdAddrTID_c2a[4:0]	Command TID
4:0	LatCmd	xxx_ctb_SampleDataIn1_x0a[4:0]	cac.lat_xxx_Command_c2a[4:0]	Command

11.11.2.4 PSx Input Collector Mux 2

Class

CtbPsxMux2

Attributes

-ocla -ctb -ctbpsx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:27	PsxToCswCmd	xxx_ctb_SampleDataIn2_x0a[31:27]	cac.psx_csw_Command_c0a[4:0]	Processor segment to cache switch command
26	CtlToSlcInvPrb	xxx_ctb_SampleDataIn2_x0a[26]	cac.ctl_slc_InvPrb_c6a	tbs
25	CtlToSlcWinPrb	xxx_ctb_SampleDataIn2_x0a[25]	cac.ctl_slc_WinPrb_c6a	tbs
24	SlcToLamRdyState1bit3	xxx_ctb_SampleDataIn2_x0a[24]	cac.slc_lam_RdyState1_c2a[3]	Bit3 of RdyState1, a mistake, but can be used
23:21	ReqEnc	xxx_ctb_SampleDataIn2_x0a[23:21]	[See_Note_1]	Encoding of which tag flag set, 0 if multiple
20	TagToCtlPrbWay	xxx_ctb_SampleDataIn2_x0a[20]	cac.tag_ctl_PrbWay_c6a	tbs
19	TagToCtlPrbHit	xxx_ctb_SampleDataIn2_x0a[19]	cac.tag_ctl_PrbHit_c6a	tbs
18:17	TagToCtlBlkState	xxx_ctb_SampleDataIn2_x0a[18:17]	cac.tag_ctl_BlkState_cza[1:0]	tbs
16	TagToCtlW1Miss	xxx_ctb_SampleDataIn2_x0a[16]	cac.tag_ctl_W1Miss_cza	See Note 2: Ice9A - Tag Miss or Idle on Way-1 (same as ~Hit) Ice9B - Tag Hit on Way-1
15	TagToCtlW0Miss	xxx_ctb_SampleDataIn2_x0a[15]	cac.tag_ctl_W0Miss_cza	See Note 2: Tag Miss or Idle on Way-0 (same as ~Hit) Ice9B - Tag Hit on Way-0
14:13	SlcPrbDirty	xxx_ctb_SampleDataIn2_x0a[14:13]	cac.slc_xxx_PrbDirty_cya[1:0]	SLC Dirty Probe
12:11	SlcPrbDone	xxx_ctb_SampleDataIn2_x0a[12:11]	cac.slc_xxx_PrbDone_cya[1:0]	SLC Probe Done
10	CtlToDatWtPrb2L2	xxx_ctb_SampleDataIn2_x0a[10]	cac.ctl_dat_WtPrb2L2_c5a	tbs
9	CtlToDatPrbRdReq	xxx_ctb_SampleDataIn2_x0a[9]	cac.ctl_dat_PrbRdReq_c5a	tbs
8	SlcBiuPaused	xxx_ctb_SampleDataIn2_x0a[8]	cac.slc_xxx_BiuPaused_c2b	tbs
7	SlcToDatPrbWbVal	xxx_ctb_SampleDataIn2_x0a[7]	cac.slc_dat_PrbWbVal_cya	tbs
6:3	SlcToLamRdyState0	xxx_ctb_SampleDataIn2_x0a[6:3]	cac.slc_lam_RdyState0_c2a[3:0]	tbs
2	SlcToTagBiuRead	xxx_ctb_SampleDataIn2_x0a[2]	cac.slc_tag_BiuRead_cya	tbs
1	SlcToTagBiuWrite	xxx_ctb_SampleDataIn2_x0a[1]	cac.slc_tag_BiuWrite_cya	tbs
0	SlcToTagBiuMemAcc	xxx_ctb_SampleDataIn2_x0a[0]	cac.slc_tag_BiuMemAcc_cya	tbs

Note 1:

case ({cac.ctl_tag_InvReq_c5a, cac.ctl_tag_WinReq_c5a, cac.ctl_tag_BrdReq_c5a, cac.ctl_tag_BwtReq_c5a, cac.ctl_tag_ShrReq_c5a})

5'b00001 : xxx_ctb_SampleDataIn2_x0a[23:21] <= 3'd1; // ShrReq

5'b00010 : xxx_ctb_SampleDataIn2_x0a[23:21] <= 3'd2; // BwtReq

5'b00100 : xxx_ctb_SampleDataIn2_x0a[23:21] <= 3'd3; // BrdReq

5'b01000 : xxx_ctb_SampleDataIn2_x0a[23:21] <= 3'd4; // WinReq

5'b10000 : xxx_ctb_SampleDataIn2_x0a[23:21] <= 3'd5; // InvReq

default : xxx_ctb_SampleDataIn2_x0a[23:21] <= 3'd0; // none of the above, or more-than-one of the above

endcase

Note 2:

In Ice9A bits 15 and 16 are cac.tag_ctl_W0Miss_cza and cac.tag_ctl_W1Miss_cza.

In Ice9B and later bits 15 and 16 are cac.tag_ctl_W0Hit_cza and cac.tag_ctl_W1Hit_cza.

11.11.2.5 PSx Input Collector Mux 3

Class

CtbPsxMux3

Attributes

-ocla -ctb -ctbpsx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:0		xxx_ctb_SampleDataIn3_x0a[31:0]	[always_zero]	Reserved

11.11.2.6 PSx Input Collector Mux 4, 5, 6, 7

The data mux leading into PSx CTBs has only the lower 2 bits of ExtMuxSel wired-up, selecting between the 4 options described above. This means ExtMuxSel values 4,5,6,7 give you the same data choices as 0,1,2,3.

11.12 OCLA in use – COHx

"COHx" means either of COHe or COHo.

11.12.0.7 COHx Trigger and Collector Enabling

Due to a minor bug affecting COHx only, both Trigger and Collector Blocks must be enabled to use either. By "enabled" I mean setting their external muxes to other than 7. COHe and COHo are separately enabled. They all default to 7, which disables OCLA activities, saving power.

For example: If all I wanted to use was the COHo Collector Block (triggering was done elsewhere, not in COH), I would need to set COHo Collector Block External Mux to the setting for what I wanted to collect, and I would need to set COHo Codeword Trigger Block External Mux to any value other than 7. COHe external muxes could be left at their default values.

11.12.1 COHx Triggers

The following tables define the codeword triggers for both the Even and Odd coherence controllers. For the ICE9, the coherence units provide up to four mux selectable groupings of codeword triggers. Each class below represents one of the four mux selectable groupings.

11.12.1.1 COHx Codeword Trigger Mux 0: Trigger on incoming command/source/data-op + tag-results + orc/wbc hit

Class

TrbcCohxMux0

Attributes

-ocla -trbc -trbccohx

Bit	Mnemonic	(Codeword Sample Input)	(COH Signals)	Definition
W0[4]	TagShr	xxx_trbc_CodeSamp0Mux0_x0a[4]	ml_TagShr_c4a	Shared tag flag
W0[3]	TagHit	xxx_trbc_CodeSamp0Mux0_x0a[3]	ml_TagHit_c4a	Tag hit flag
W0[2:0]	Owner	xxx_trbc_CodeSamp0Mux0_x0a[2:0]	ml_Owner_c4a[2:0]	Tag owner mask
W1[4]	Always0	xxx_trbc_CodeSamp1Mux0_x0a[3]	unused	Hardwired to logic '0'
W1[3]	CohToDdrRdShootDwn	xxx_trbc_CodeSamp1Mux0_x0a[3]	m_coh_ddr_RdShootDown_c5a || m_RaWShootDown_c4a	tbs
W1[2]	WbcToCtlAddrHit	xxx_trbc_CodeSamp1Mux0_x0a[2]	m_wbc_ctl_AddrHit_c4a	tbs
W1[1]	OrcToCtlAddrHit	xxx_trbc_CodeSamp1Mux0_x0a[1]	m_orc_ctl_AddrHit_c4a	tbs
W1[0]	VicVal	xxx_trbc_CodeSamp1Mux0_x0a[0]	ml_VicVal_c4a	tbs
W2[4:0]	CmdAddrTid	xxx_trbc_CodeSamp2Mux0_x0a[4:0]	m_InCmdAddrTID_c3a[4:0]	Inbound Command TID
W3[4:0]	Cmd	xxx_trbc_CodeSamp3Mux0_x0a[4:0]	m_InCommand_c3a[4:0]	Inbound Command
W4[0]	Cv0Always1	xxx_trbc_CodeValid0_x0a	Hardwired to logic '1'	Hardwired to '1'
W5[0]	Cv1InCmdAddrVal	xxx_trbc_CodeValid1_x0a	m_cmd_xxx_InCmdAddrValid_c3a	tbs

Note that W0 signals are delayed by 1 cclk compared with InCmdAddrValid and other signals, and W1 signals are delayed by 2 cclks compared with InCmdAddrValid and other signals.

11.12.1.2 COHx Codeword Trigger Mux 1: Trigger on ORC/WBC behavior + incoming command

Class

TrbcCohxMux1

Attributes

-ocla -trbc -trbccohx

Bit	Mnemonic	(Codeword Sample Input)	(COH Signals)	Definition
W0[4]	WbcToCtlWrsHit	xxx_trbc_CodeSamp0Mux1_x0a[4]	m_wbc_ctl_WrsHit_c7a	Dependent share in the WBC
W0[3]	WbcToCtlDepShr	xxx_trbc_CodeSamp0Mux1_x0a[3]	m_wbc_ctl_DepShr_c5a	Dependent share in the WBC
W0[2]	WbcToCtlDepVal	xxx_trbc_CodeSamp0Mux1_x0a[2]	m_wbc_ctl_DepVal_c5a	Dependent value in the WBC
W0[1]	OrcToCtlPrbHit	xxx_trbc_CodeSamp0Mux1_x0a[1]	m_orc_ctl_PrbHit_c4a	Probe hit flag in the ORC
W0[0]	OrcToCltDdrHit	xxx_trbc_CodeSamp0Mux1_x0a[0]	m_orc_ctl_DDRHit_c12a	DDR RAM hit flag in the ORC
W1[4:0]	OrcToCtlTid	xxx_trbc_CodeSamp1Mux1_x0a[4:0]	[See_Note_1]	TID based on hit or dep value
W2[4:0]	CmdAddrTid	xxx_trbc_CodeSamp2Mux1_x0a[4:0]	m_InCmdAddrTID_c3a	Inbound command TID
W3[4:0]	Cmd	xxx_trbc_CodeSamp3Mux1_x0a[4:0]	m_InCommand_c3a	Inbound command
W4[0]	Cv0Always1	xxx_trbc_CodeValid0_x0a	Hardwired to logic '1'	Hardwired to '1'
W5[0]	Cv1InCmdAddrValid	xxx_trbc_CodeValid1_x0a	m_cmd_xxx_InCmdAddrValid_c3a	Inbound command address-valid

Notes:

1. (orc_ctl_DDRHit_c12a ? orc_ctl_DDRDepTIDc12a : 0) | (orc_ctl_PrbHit_c4a ? orc_ctlPrbDepTID_c4a : 0) | (orc_ctl_DepVal_c5a ? wbc_ctl_DepTID_c5a : 0) | (wbc_ctl_WrsHit_c7a ? wbc_ctl_WrsTID_c7a : 0) | (wbc_ctl_BwtCanHit_c4a ? wbc_ctl_BwtCanDepTID_c : 0)

11.12.1.3 COHx Codeword Trigger Mux 2: Trigger on the DDR Interface

Class

TrbcCohxMux2

Attributes

-ocla -trbc -trbccohx

Bit	Mnemonic	(Codeword Sample Input)	(COH Signals)	Definition
W0[4]	Always0	xxx_trbc_CodeSamp0Mux2_x0a[4]	unused	Hardwired to logic '0'
W0[3]	CohToDdrRdShootDwn	xxx_trbc_CodeSamp0Mux2_x0a[3]	[See_Note_1]	tbs
W0[2]	DdrToCohWtTidVal	xxx_trbc_CodeSamp0Mux2_x0a[2]	m_ddr_coh_WtTIDVal_c6a	tbs
W0[1]	DdrToCohRdShotDown	xxx_trbc_CodeSamp0Mux2_x0a[1]	m_ddr_coh_RdShotDown_c3a	tbs
W0[0]	DdrToCohDataValid	xxx_trbc_CodeSamp0Mux2_x0a[0]	m_ddr_coh_DataValid_c3a	tbs
W1[4:0]	DdrToCohTid	xxx_trbc_CodeSamp1Mux2_x0a[4:0]	[See_Note_2]	tbs
W2[4:0]	CohToDdrWrTid	xxx_trbc_CodeSamp2Mux2_x0a[4:0]	[See_Note_3]	tbs
W3[4:0]	CohRdTid	xxx_trbc_CodeSamp3Mux2_x0a[4:0]	[See_Note_4]	tbs
W4[0]	Cv0Always1	xxx_trbc_CodeValid0_x0a	Hardwired to logic '1'	tbs
W5[0]	Cv1InCmdAddrValid	xxx_trbc_CodeValid1_x0a	m_cmd_xxx_InCmdAddrValid_c3a	tbs

Notes:

1. m_coh_ddr_RdShootDown_c5a || m_coh_ddr_RaWShootDown_c4a
2. ((m_ddr_coh_DataValid_c3a || m_ddr_coh_RdShotDown_c3a) ? m_ddr_coh_DataTID_c3a : 0x00) | (m_ddr_coh_WtTIDVal_c6a ? m_ddr_coh_WtTID_c6a : 0x00)
3. m_coh_ddr_WrValid_c6a ? m_coh_ddr_WrTID_c6a : 0x1f
4. m_cohddr_RdValid_c3a ? m_cohddr_RdTID_c3a : 0x1f

11.12.1.4 COHx Codeword Trigger Mux 3: Trigger on an Incoming Address

Class

TrbcCohxMux3

Attributes

-ocla -trbc -trbccohx

Bit	Mnemonic	(Codeword Sample Input)	(COH Signals)	Definition
W0[4:0]	InAddr	xxx_trbc_CodeSamp0Mux3[4:0]	m_cmd_xxx_InAddr_c3a[8:7], m_cmd_xxx_InAddr_c3a[5:3]	Incoming ? Address
W1[4:0]	InPageAddr	xxx_trbc_CodeSamp1Mux3[4:0]	m_cmd_xxx_InAddr_c3a[20:16]	Incoming Page Address
W2[4:0]	OutRdAddr1	xxx_trbc_CodeSamp2Mux3[4:0]	m_cohddr_RdAddr_c3a[8:7], m_cohddr_RdAddr_c3a[5:3]	Outgoing ? Address
W3[4:0]	OutRdAddr2	xxx_trbc_CodeSamp3Mux3[4:0]	m_cohddr_RdAddr_c3a[8:7], m_cohddr_RdAddr_c3a[5:3]	(Same as mux selection 2)
W4[0]	Cv0Always1	xxx_trbc_CodeValid0_x0a	Hardwired to logic '1'	tbs
W5[0]	Cv1InCmdAddrValid	xxx_trbc_CodeValid1_x0a	m_cmd_xxx_InCmdAddrValid_c3a	tbs

11.12.2 COHx Collectors

Each of the COHx units, Cohe (even) and Coho (odd), will have a collector to record commands, TIDs, and tag indices arriving at that COH.

Note: If you are looking at the COH source code, you'll see 4 OCLA collectors instantiated in Cohe and 4 in Coho! These are 1 for the COHx unit, and 3 for PSx units. When using OCLA, you don't have to pay attention to where the collectors are actaully instantiated, all you care about is what signals they're hooked to. So, for functional purposes, each COHx has only 1 OCLA Collector.

11.12.2.1 Cohx Input Collectors Qualifying Triggers

Class

CtbCohxQtrig

Attributes

-ocla -ctb -ctbcohx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
1	InCmdAddrValid	xxx_ctb_QualTrigger1_x0a	m_cmd_xxx_InCmdAddrValid_c3a	tbs
0	OutTarget	xxx_ctb_QualTrigger0_x0a	[See_Note_1]	tbs

Notes:

1. m_coh_csw_OutDataTarget_c3a[0] || m_coh_csw_OutCmdAddrTarget_c1a[0]

11.12.2.2 Cohx Input Collector Mux 0

Class

CtbCohxMux0

Attributes

-ocla -ctb -ctbcohx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:16	InAddrHi	xxx_ctb_SampleDataIn0_x0a[31:16]	m_InAddr_c3a[31:16]	Page Address [31:16]

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
15:13	InAddrLo	xxx_ctb_SampleDataIn0_x0a[15:13]	m_InAddr_c3a[5:3]	Page Address [5:3]
12:8	InCmd	xxx_ctb_SampleDataIn0_x0a[12:8]	m_InCommand_c3a[4:0]	Incomming command
7:3	InCmdAddrTid	xxx_ctb_SampleDataIn0_x0a[7:3]	m_InCmdAddrTID_c3a[4:0]	Incomming TID
2:0	Owner	xxx_ctb_SampleDataIn0_x0a[2:0]	ml_Owner_c4a[2:0]	Block Owner

11.12.2.3 Cohx Input Collector Mux 1

Class

CtbCohxMux1

Attributes

-ocla -ctb -ctbcohx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31	OrcToCtlAddrHit	xxx_ctb_SampleDataIn1_x0a[31]	m_orc_ctl_AddrHit_c4a	ORC Cache Address Hit
30	WbcToCtlAddrHit	xxx_ctb_SampleDataIn1_x0a[30]	m_wbc_ctl_AddrHit_c4a	Write Back Cache Address Hit
29	OrcToCtlDdrHit	xxx_ctb_SampleDataIn1_x0a[29]	m_orc_ctl_DDRHit_c12a	DDR Hit
28	OrcToCtlPrbHit	xxx_ctb_SampleDataIn1_x0a[28]	m_orc_ctl_PrbHit_c4a	Cache Probe Hit
27	WbcToCtlDepVal	xxx_ctb_SampleDataIn1_x0a[27]	m_wbc_ctl_DepVal_c5a	tbs
26	WbcToCtlDepShr	xxx_ctb_SampleDataIn1_x0a[26]	m_wbc_ctl_DepShr_c5a	tbs
25	WbcToCtlWrsHit	xxx_ctb_SampleDataIn1_x0a[25]	m_wbc_ctl_WrsHit_c7a	tbs
24	WbcToCtlBwtCanHit	xxx_ctb_SampleDataIn1_x0a[24]	m_wbc_ctl_BwtCanHit_c4a	tbs
23:19	DepTid	xxx_ctb_SampleDataIn1_x0a[23:19]	ml_DepTID_ca[See_Note_1]	tbs
18	DdrToCohDvOrRdShtDwn	xxx_ctb_SampleDataIn1_x0a[18]	[See_Note_2]	tbs
17	CohToDdrRdShootDwn	xxx_ctb_SampleDataIn1_x0a[17]	[See_Note_3]	tbs
16:13	DdrToCohDataTid	xxx_ctb_SampleDataIn1_x0a[16:13]	m_ddr_coh_DataTID_c3a[4:1]	tbs
12:8	InCmd	xxx_ctb_SampleDataIn1_x0a[12:8]	m_cmd_xxx_InCommand_c3a[4:0]	Incoming command
7:3	InCmdAddrTid	xxx_ctb_SampleDataIn1_x0a[7:3]	m_cmd_xxx_InCmdAddrTID_c3a[4:0]	Incoming command TID
2:0	Owner	xxx_ctb_SampleDataIn1_x0a[2:0]	ml_Owner_c4a[2:0]	tbs

Notes:

1. ml_DepTID_ca =
   (a) (m_orc_ctl_DDRHit_c12a ? m_orc_ctl_DDRDepTID_c12a : 0x00)
   (b) \| (m_orc_ctl_PrbHit_c4a ? m_orc_ctl_PrbDepTID_c4a : 0x00)
   (c) \| (m_wbc_ctl_DepVal_c5a ? m_wbc_ctl_DepTID_c5a : 0x00)
   (d) \| (m_wbc_ctl_WrsHit_c7a ? m_wbc_ctl_WrsTID_c7a : 0x00)
   (e) \| (m_wbc_ctl_BwtCanHit_c4a ? m_wbc_ctl_BwtCanDepTID_c4a : 0x00);
2. (m_ddr_coh_DataValid_c3a \|\| m_ddr_coh_RdShotDown_c3a)
3. (m_coh_ddr_RdShootDown_c5a \|\| m_coh_ddr_RaWShootDown_c4a)

11.12.2.4 Cohx Input Collector Mux 2

Class

CtbCohxMux2

Attributes

-ocla -ctb -ctbcohx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:27	CohToCswOutCmdAddrTid	xxx_ctb_SampleDataIn2_x0a[31:27]	m_coh_csw_OutCmdAddrTID_c1a[4:0]	tbs
26:22	CohToCswOutCmd	xxx_ctb_SampleDataIn2_x0a[26:22]	[See_Note_1]	tbs
21:18	CohToCswOutCmdOrigin	xxx_ctb_SampleDataIn2_x0a[21:18]	m_coh_csw_OutCmdOrigin_c1a[3:0]	tbs
17:13	CohToCswOutDataTid	xxx_ctb_SampleDataIn2_x0a[17:13]	[See_Note_2]	tbs
12:8	InCmd	xxx_ctb_SampleDataIn2_x0a[12:8]	m_cmd_xxx_InCommand_c3a[4:0]	tbs
7:3	InCmdAddrTid	xxx_ctb_SampleDataIn2_x0a[7:3]	m_cmd_xxx_InCmdAddrTID_c3a[4:0]	tbs
2:0	Owner	xxx_ctb_SampleDataIn2_x0a[2:0]	ml_Owner_c4a[2:0]	tbs

Notes:

1. ((m_coh_csw_OutCmdAddrTarget_c1a != 0x000) ? m_coh_csw_OutCommand_c1a : E_CohCmd_IDLE) where E_CohCmd_IDLE = 0x07
2. (m_coh_csw_OutDataTarget_c3a != 0x000) ? m_coh_csw_OutDataTID_c3a : 0x1f

11.12.2.5 Cohx Input Collector Mux 3

Class

CtbCohxMux3

Attributes

-ocla -ctb -ctbcohx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:0	InAddrDW	xxx_ctb_SampleDataIn3_x0a[31:0]	m_cmd_xxx_InAddr_c3a[35:4]	Full physical address [35:4]

11.12.2.6 Cohx Input Collector Mux 4

Class

CtbCohxMux4

Attributes

-ocla -ctb -ctbcohx

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:0	CycCtr	xxx_ctb_SampleDataIn4_x0a[31:0]	m_FreeRunCtr_x0a[31:0]	A free running 32 bit counter that increments every CCLK cycle. Counting is not affected by mux selections or enabling of OCLA. Not settable. Cleared during reset. Rolls over.

This allows you to time-stamp collections within the rollover time of 2**32 cclks. This can be used in parallel with ***any*** other collector block. Since this uses either the COHe or COHo collector block, you cannot collect signals in both COHe and COHo and get these timestamps all at once.

In the COHe or COHo used, make sure to set both Collector and Trigger external muxes to non-7 values, even if no COH triggers are needed, otherwise this collector remains disabled.

Since 2**32 cclks has probably occured many times since un-reset, the usefulness of this is limited to relative times between two or more periods of collection driven from a LAC program. If your LAC program collects, then stops collecting, then starts collecting, then stops collecting, the values stored from this counter can tell you how long that middle time-period of not-collecting was. This can show you the time between two events, if you are confident that less time than 2**32 cclks has passed. One way to be sure only a short time passed is to program LAC with one of it's counters as a time-out on the middle-non-collecting time period. Another way to be sure less than 2**32 cclks have passed is for whatever processor code starts the LAC program and then checks for "done" flags, to read it's own CPU internal cycle counter while polling for "done", or just have a software timeout on polling for "done".

11.12.2.7 Cohx Input Collector Mux 5, or 6

Collect all zeros.

11.12.2.8 Cohx Input Collector Mux 7

Disable CTB.

11.13 OCLA in use – FSW

11.13.1 FSW Triggers

We'd like to be able to trigger on different events occuring at the FSW input and output ports. However, the FSW has three in/out ports from the DMA engine and three more in/out ports to the fabric link logic. That's way too much stuff to be recording and hooking on to. So we instrument DMA to FSW port-0, FSW to DMA port-0, FLR to FSW port-0, and FSW to FLT port-0.

There are three trigger units. These trigger units give us the ability to detecting start of packet/end of packet events, transitions to and from mission mode, poisoned packets and interesting routes. Trigger inputs from the control signals are routed to a Codeword TRB (TRBC) as shown in section 11.13.1.1. Four groups of 32 bits from the input data paths are routed to one Vector TRB, while four groups of 32 bits from the output data paths are routed to a second Vector TRB. Control paths to and from the links are also routed to these Vector TRBs. The Vector TRB connections are described in sections 11.13.1.2 to 11.13.1.11.

11.13.1.1 FSW Codeword Trigger Block Inputs

The Fabric Switch codeword trigger blocks define sets of events that can be enabled separately or grouped together to provide interesting triggers for events within the Fabric Switch (FSW).

Class

TrbcFsw

Attributes

-ocla -trbc -trbcfsw

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4]	FlrToFswDatVal	xxx_trbc_CodeSamp0[4]	flr0_fsw_DatVal_s0a	Data Packets Data-Valid from FLR-0
W0[3]	FlrToFswSop	xxx_trbc_CodeSamp0[3]	flr0_fsw_SoP_s0a	Start of Data Packet from FLR-0
W0[2]	FlrToFswSopD1	xxx_trbc_CodeSamp0[2]	ocla_flr_fsw_sop_d1	Start of Data Packet from FLR-0, delayed 1 sclk
W0[1]	FlrToFswEop	xxx_trbc_CodeSamp0[1]	flr0_fsw_EoP_s0a	End of Data Packet from FLR-0
W0[0]	FswToFlrNewCtlPktD1	xxx_trbc_CodeSamp0[0]	ocla_fsw_flr_newctlpkt_d1	Start of Control Packet to FLR-0, delayed 1 sclk
W1[4]	FltToFswDatVal	xxx_trbc_CodeSamp1[4]	flt0_fsw_DatVal_s0a	Control Packets Data-Valid from FLT-0
W1[3]	FswToFltSop	xxx_trbc_CodeSamp1[3]	fsw_flt0_SoP_s2a	Start of Data Packet to FLT-0
W1[2]	FswToFltSopD1	xxx_trbc_CodeSamp1[2]	ocla_fsw_flt_sop_d1	Start of Data Packet to FLT-0, delayed 1 sclk
W1[1]	FswToFltEop	xxx_trbc_CodeSamp1[1]	fsw_flt0_EoP_s2a	End of Data Packet to FLT-0
W1[0]	FltToFswNewCtlPktD1	xxx_trbc_CodeSamp1[0]	ocla_flt_fsw_newctlpkt_d1	Start of Control Packet from FLT-0, delayed 1 sclk
W2[4]	DmaToFswSop	xxx_trbc_CodeSamp2[4]	dma_fsw_SoP0_s0a	Start of Packet from DMA port TX0
W2[3]	DmaToFswSopD1	xxx_trbc_CodeSamp2[3]	ocla_dma_fsw_sop_d1	Start of Packet from DMA port TX0, delayed 1 sclk
W2[2]	DmaToFswSopD2	xxx_trbc_CodeSamp2[2]	ocla_dma_fsw_sop_d2	Start of packet from DMA port TX0, delayed 2 sclks
W2[1]	DmaToFswEop	xxx_trbc_CodeSamp2[1]	dma_fsw_EoP0_s0a	End of packet from DMA port TX0

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W2[0]	FswToDmaBufAvail	xxx_trbc_CodeSamp2[0]	fsw_dma_BufAvail0_s3a	FSW Buffer Available signal to DMA port TX0
W3[4]	FswToDmaSop	xxx_trbc_CodeSamp3[4]	fsw_dma_SoP0_s2a	Start of Packet to DMA port RX0
W3[3]	FswToDmaSopD1	xxx_trbc_CodeSamp3[3]	ocla_fsw_dma_sop_d1	Start of Packet to DMA port RX0, delayed 1 sclk
W3[2]	FswToDmaSopD2	xxx_trbc_CodeSamp3[2]	ocla_fsw_dma_sop_d2	Start of Packet to DMA port RX0, delayed 2 sclks
W3[1]	FswToDmaEop	xxx_trbc_CodeSamp3[1]	fsw_dma_EoP0_s2a	End of Packet to DMA port RX0
W3[0]	DmaToFswRdy	xxx_trbc_CodeSamp3[0]	dma_fsw_Rdy0_s1a	DMA ready for new packet from FSW on port RX0
W4[0]	Cv0FlrToFswMsnMode	xxx_trbc_CodeValid0	flr0_fsw_MissionMode	MissionMode from FLR-0
W5[0]	Cv1FltToFswMsnMode	xxx_trbc_CodeValid1	flt0_fsw_MissionMode	MissionMode from FLT-0

11.13.1.2 FSW Input Vector Trigger (Mux 0)

These are the fields selected from data coming into the FSW when MuxSel=0.

Class

TrbvFswiMux0

Attributes

-ocla -trbv -trbvfswi

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FlrToFswInDat	flr0_fsw_InDat_s0a[63:60], flr0_fsw_InDat_s0a[35:8]	Fields selected for data coming into the FSW.
W1[0]	FlrToFswIdle	flr0_fsw_Idle_s0a	Data from link is IDLE packet or Data packet

11.13.1.3 FSW Input Vector Trigger (Mux 1)

These are the fields selected from data coming into the FSW when MuxSel=1.

Class

TrbvFswiMux1

Attributes

-ocla -trbv -trbvfswi

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FlrToFswInDat	flr0_fsw_InDat_s0a[59:36], flr0_fsw_InDat_s0a[7:0]	Fields selected for data coming into the FSW.
W1[0]	FlrToFswIdle	flr0_fsw_Idle_s0a	Data from link is IDLE packet or DATA packet

11.13.1.4 FSW Input Vector Trigger Mux 2

These are the fields selected from data coming into the FSW when MuxSel=2.

Class

TrbvFswiMux2

Attributes

-ocla -trbv -trbvfswi

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FswToFlrCtlDat	flr0_fsw_InDat_s0a[59:36], fsw_flr0_CtlDat_s3a[7:0]	Fields selected for data coming into the FSW.
W1[0]	FlrToFswIdle	flr0_fsw_Idle_s0a	Data from link is IDLE packet or DATA packet

Although fsw_flr0_CtlDat_s3a[7:0] is an output of FSW, it's considered part of the "FLR0 input interface" to FSW, so we provide it as an option in the FSW Input trigger block.

11.13.1.5 FSW Input Vector Trigger Mux 3

These are the fields selected from data coming into the FSW when MuxSel=3.

Class

TrbvFswiMux3

Attributes

-ocla -trbv -trbvfswi

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	DmaToFswInDat	dma_fsw_InDat0_s0a[63:60], dma_fsw_InDat0_s0a[35:8]	Fields selected for data coming into the FSW.
W1[0]	DmaToFswDatVal	dma_fsw_DatVal0_s0a	Data from DMA engine is worth looking at

11.13.1.6 FSW Input Vector Trigger Mux 4

These are the fields selected from data coming into the FSW when MuxSel=4.

Class

TrbvFswiMux4

Attributes

-ocla -trbv -trbvfswi

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	DmaToFswInDat	dma_fsw_InDat0_s0a[59:36], dma_fsw_InDat0_s0a[7:0]	Fields selected for data coming into the FSW.
W1[0]	DmaToFswDatVal	dma_fsw_DatVal0_s0a	Data from DMA engine is worth looking at

11.13.1.7 FSW Output Vector Trigger Mux 0

These are the fields selected from data being driven from the FSW when MuxSel=0.

Class

TrbvFswoMux0

Attributes

-ocla -trbv -trbvfswo

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FswToFltOutDat	fsw_flt0_OutDat_s2a[63:60], fsw_flt0_OutDat_s2a[35:8]	tbs
W1[0]	FswToFltIdle	fsw_flt0_Idle_s2a	Data from link is IDLE packet or DATA packet

11.13.1.8 FSW Output Vector Trigger Mux 1

These are the fields selected from data being driven from the FSW when MuxSel=1.

Class

TrbvFswoMux1

Attributes

-ocla -trbv -trbvfswo

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FswToFltOutDat	fsw_flt0_OutDat_s2a[59:36], fsw_flt0_OutDat_s2a[7:0]	tbs
W1[0]	FswToFltIdle	fsw_flt0_Idle_s2a	Data from link is IDLE packet or DATA packet

11.13.1.9 FSW Output Vector Trigger Mux 2

These are the fields selected from data being driven from the FSW when MuxSel=2.

Class

TrbvFswoMux2

Attributes

-ocla -trbv -trbvfswo

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FltToFswCtlDat	fsw_flt0_OutDat_s2a[59:36], flt0_fsw_CtlDat_s0a[7:0]	tbs
W1[0]	FswToFltIdle	fsw_flt0_Idle_s2a	Data from link is IDLE packet or DATA packet

Although flt0_fsw_fsw_CtlDat_s0a[7:0] is an input of FSW, it's considered part of the "FLT0 output interface" to FSW, so we provide it as an option in the FSW Output trigger block.

11.13.1.10 FSW Output Vector Trigger Mux 3

These are the fields selected from data being driven from the FSW when MuxSel=3.

Class

TrbvFswoMux3

Attributes

-ocla -trbv -trbvfswo

Bit	Mnemonic	(Signal)	Definition

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FswToDmaOutDat	fsw_dma_OutDat0_s2a[63:60],tbs fsw_dma_OutDat0_s2a[35:8]	
W1[0]	FswToDmaDatVal	fsw_dma_DatVal0_s2a	Data from DMA engine is worth looking at

11.13.1.11 FSW Output Vector Trigger Mux 4

These are the fields selected from data being driven from the FSW when MuxSel=4.

Class

TrbvFswoMux4

Attributes

-ocla -trbv -trbvfswo

Bit	Mnemonic	(Signal)	Definition
W0[31:0]	FswToDmaOutDat	fsw_dma_OutDat0_s2a[59:36],tbs fsw_dma_OutDat0_s2a[7:0]	
W1[0]	FswToDmaDatVal	fsw_dma_DatVal0_s2a	Data from DMA engine is worth looking at

11.13.2 FSW Collectors

The FSW contains two CTBs, one for incoming data and one for outgoing data. The CTB for incoming data is connected to the same signals as the FSW Input Vector Trigger Block. The CTB for outgoing data is connected to the same signals as the FSW Output Vector Trigger Block.

11.13.2.1 FSW Input Collectors Qualifying Triggers

Class

CtbFswiQtrig

Attributes

-ocla -ctb -ctbfswi

Bit	Mnemonic	(Signal)	Definition
1	DmaToFswDatVal	fsw.dma_fsw_DatVal0_s0a	Qualify collection on Dma to Fsw data valid.
0	FlrToFswIdle	fsw.flr0_fsw_Idle_s0a	Qualify collection on Flr0 to Fsw Idle.

11.13.2.2 FSW Input Collector Mux 0

Class

CtbFswiMux0

Attributes

-ocla -ctb -ctbfswi

Bit	Mnemonic	(Signal)	Definition
31:28	FlrToFswDat6360	fsw.flr0_fsw_InDat_s0a[63:60]	Data from FLR0 to FSW bits 63-60.
27:0	FlrToFswDat358	fsw.flr0_fsw_InDat_s0a[35:8]	Data from FLR0 to FSW bits 35-8.

11.13.2.3 FSW Input Collector Mux 1

Class

CtbFswiMux1

Attributes

-ocla -ctb -ctbfswi

Bit	Mnemonic	(Signal)	Definition
31:8	FlrToFswDat5936	fsw.flr0_fsw_InDat_s0a[59:36]	Data from FLR0 to FSW bits 59-36.
7:0	FlrToFswDat70	fsw.flr0_fsw_InDat_s0a[7:0]	Data from FLR0 to FSW bits 7-0.

11.13.2.4 FSW Input Collector Mux 2

Class

CtbFswiMux2

Attributes

-ocla -ctb -ctbfswi

Bit	Mnemonic	(Signal)	Definition
31:8	FlrToFswDat5936	fsw.flr0_fsw_InDat_s0a[59:36]	Data from FLR0 to FSW bits 59-36.
7:0	FswToFlrCtlDat	fsw.fsw_flr0_CtlDat_s3a[7:0]	Control Data from FSW to FLR0.

Although fsw_flr0_fsw_CtlDat_s3a[7:0] is an output of FSW, it's considered part of the "FLR0 input interface" to FSW, so we provide it as an option in the FSW Input collector block.

11.13.2.5 FSW Input Collector Mux 3

Class

CtbFswiMux3

Attributes

-ocla -ctb -ctbfswi

Bit	Mnemonic	(Signal)	Definition
31:28	DmaToFswDat6360	fsw.dma_fsw_InDat0_s0a[63:60]	Data from DMA to FSW bits 63-60.
27:0	DmaToFswDat358	fsw.dma_fsw_InDat0_s0a[35:8]	Data from DMA to FSW bits 35-8.

11.13.2.6 FSW Input Collector Mux 4

Class

CtbFswiMux4

Attributes

-ocla -ctb -ctbfswi

Bit	Mnemonic	(Signal)	Definition

Bit	Mnemonic	(Signal)	Definition
31:8	DmaToFswDat5936	fsw.dma_fsw_InDat0_s0a[59:36]	Data from DMA to FSW bits 59-36.
7:0	DmaToFswDat70	fsw.dma_fsw_InDat0_s0a[7:0]	Data from DMA to FSW bits 7-0.

11.13.2.7 FSW Input Collector Mux 5, 6, 7

Gives you the same as Mux 4.

11.13.2.8 FSW Output Collectors Qualifying Triggers

Class

CtbFswoQtrig

Attributes

-ocla -ctb -ctbfswo

Bit	Mnemonic	(Signal)	Definition
1	FswToDmaDatVal	fsw.fsw_dma_DatVal0_s2a	Qualify collection on Fsw to Dma data valid.
0	FswToFltIdle	fsw.fsw_flt0_Idle_s2a	Qualify collection on Fsw to Flt0 Idle.

11.13.2.9 FSW Output Collector Mux 0

Class

CtbFswoMux0

Attributes

-ocla -ctb -ctbfswo

Bit	Mnemonic	(Signal)	Definition
31:28	FswToFltDat6360	fsw.fsw_flt0_OutDat_s2a[63:60]	Data from FSW to FLT0 bits 63-60.
27:0	FswToFltDat358	fsw.fsw_flt0_OutDat_s2a[35:8]	Data from FSW to FLT0 bits 35-8.

11.13.2.10 FSW Output Collector Mux 1

Class

CtbFswoMux1

Attributes

-ocla -ctb -ctbfswo

Bit	Mnemonic	(Signal)	Definition
31:8	FswToFltDat5936	fsw.fsw_flt0_OutDat_s2a[59:36]	Data from FSW to FLT0 bits 59-36.
7:0	FswToFltDat70	fsw.fsw_flt0_OutDat_s2a[7:0]	Data from FSW to FLT0 bits 7-0.

11.13.2.11 FSW Output Collector Mux 2

Class

CtbFswoMux2

Attributes

-ocla -ctb -ctbfswo

Bit	Mnemonic	(Signal)	Definition
31:8	FswToFltDat5936	fsw.fsw_flt0_OutDat_s2a[59:36]	Data from FSW to FLT0 bits 59-36.
7:0	FltToFswCtlDat	fsw.flt0_fsw_CtlDat_s0a[7:0]	Control Data from FLT0 to FSW.

Although flt0_fsw_fsw_CtlDat_s0a[7:0] is an input of FSW, it's considered part of the "FLT0 output interface" to FSW, so we provide it as an option in the FSW Output collector block.

11.13.2.12 FSW Output Collector Mux 3

Class

CtbFswoMux3

Attributes

-ocla -ctb -ctbfswo

Bit	Mnemonic	(Signal)	Definition
31:28	FswToDmaDat6360	fsw.fsw_dma_OutDat0_s2a[63:60]	Data from FSW to DMA bits 63-60.
27:0	FswToDmaDat358	fsw.fsw_dma_OutDat0_s2a[35:8]	Data from FSW to DMA bits 35-8.

11.13.2.13 FSW Output Collector Mux 4

Class

CtbFswoMux4

Attributes

-ocla -ctb -ctbfswo

Bit	Mnemonic	(Signal)	Definition
31:8	FswToDmaDat5936	fsw.fsw_dma_OutDat0_s2a[59:36]	Data from FSW to DMA bits 59-36.
7:0	FswToDmaDat70	fsw.fsw_dma_OutDat0_s2a[7:0]	Data from FSW to DMA bits 7-0.

11.13.2.14 FSW Output Collector Mux 5, 6, 7

Gives you the same as Mux 4.

11.14 OCLA in use – DMA

11.14.1 DMA Triggers

The DMA engine has a CSW Bus Stop trigger and collector unit, one vector trigger unit and one capture block. The inputs to the TRBV and the CTB are muxed from a set of 128 signals. The CSW side of the DMA engine is connected to a TRBC unit with connections shown in Section 11.14.1.1.

11.14.1.1 DMA Codeword Triggers

DMA Engine to Central Switch codeword triggers.

Class

TrbcDma

Attributes

-ocla -trbc -trbcdma

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4:0]	CswToDmaCmd	xxx_trb_CodeSamp0[4:0]	m_csw_dma_Command_c2a[4:0]	The incoming command code
W1[4:0]	CswToDmaCmdAddrTid	xxx_trb_CodeSamp1[4:0]	m_csw_dma_CmdAddrTID_c2a[4:0]	The incoming command TID
W2[4:0]	CswToDmaDataTid	xxx_trb_CodeSamp2[4:0]	m_csw_dma_DataTID_c4a[4:0]	The incoming data TID
W3[4]		xxx_trb_CodeSamp3[4]		Reserved (drive to '0')
W3[3]		xxx_trb_CodeSamp3[3]		Reserved (drive to '0')
W3[2]	DmaToCswECmdAddrReq	xxx_trb_CodeSamp3[2]	dma_csw_ECmdAddrReq_c1a	Even bound command request
W3[1]	DmaToCswOCmdAddrReq	xxx_trb_CodeSamp3[1]	dma_csw_OCmdAddrReq_c1a	Odd bound command request
W3[0]	CswToDmaCmdAddrGnt	xxx_trb_CodeSamp3[0]	csw_dma_CmdAddrGnt_c2a	Comand grant
W4[0]	Cv0CswToDmaCmdAddrValid	xxx_ctb_CodeValid0_x0a	m_csw_dma_CmdAddrValid_c2a	Comand/transfer is valid
W5[0]	Cv1CswToDmaDataValid	xxx_ctb_CodeValid1_x0a	m_csw_dma_DataValid_c4a	Data is valid

The input to the TRBV is selected as shown in Sections 11.14.1.2, 11.14.1.3, 11.14.1.4, and 11.14.1.5. The TRBV trb_xxx_MuxSel_xa[1:0] outputs select from among the four groups. The TRBV has one CodeValid input, connected to m_ue_xxx_DbgValid_c2a.

11.14.1.2 DMA Vector Trigger Inputs (Mux 0)

DMA Engine transmit and receive port buffer status.

Class

TrbvDmaMux0

Attributes

-ocla -trbv -trbvdma

Bit	Mnemonic	(Signal)	Definition
W0[31]	Rxp0ToUEngBufAvail	rxp0_ue_BufAvail_c1a	Receive port 0 to microengine buffer available
W0[30]	Rxp1ToUEngBufAvail	rxp1_ue_BufAvail_c1a	Receive port 1 to microengine buffer available
W0[29]	Rxp2ToUEngBufAvail	rxp2_ue_BufAvail_c1a	Receive port 2 to microengine buffer available
W0[28]	UEngRxThreadStart	copy_ue_RxThreadStart_c1a	Microengine receive thread start
W0[27]	Txp0ToUEngBufAvail	txp0_ue_BufAvail_c1a	Transmit port 0 to microengine buffer available
W0[26]	Txp1ToUEngBufAvail	txp1_ue_BufAvail_c1a	Transmit port 1 to microengine buffer available
W0[25]	Txp2ToUEngBufAvail	txp2_ue_BufAvail_c1a	Transmit port 2 to microengine buffer available
W0[24]	UEngTxThreadStart	copy_ue_TxThreadStart_c1a	Microengine transmit thread start
W0[23]	UEngToRxp0BufXfr	ue_rxp0_BufTransfer_c5a	Microengine to receive port 0 buffer transfer
W0[22]	UEngToRxp1BufXfr	ue_rxp1_BufTransfer_c5a	Microengine to receive port 1 buffer transfer
W0[21]	UEngToRxp2BugXfr	ue_rxp2_BufTransfer_c5a	Microengine to receive port 2 buffer transfer

Bit	Mnemonic	(Signal)	Definition
W0[20]	UEngRxThreadDone	ue_copy_RxThreadDone_c5a	Microengine receive thread done
W0[19]	UEngToTxp0BufXfr	ue_txp0_BufTransfer_c5a	Microengine to transmit port 0 buffer transfer
W0[18]	UEngToTxp1BufXfr	ue_txp1_BufTransfer_c5a	Microengine to transmit port 1 buffer transfer
W0[17]	UEngToTxp2BufXfr	ue_txp2_BufTransfer_c5a	Microengine to transmit port 2 buffer transfer
W0[16]	UEngTxThreadDone	ue_copy_TxThreadDone_c5a	Microengine transmit thread done
W0[15]		unused	Reserved
W0[14]	UEngDbgThreadValid	ue_xxx_DbgValid_c4a	Microengine thread valid
W0[13:10]	UEngDbgThread	ue_xxx_DbgThread_c4a[3:0]	Microengine thread number
W0[9:0]	UEngDbgPc	ue_xxx_DbgPc_c4a[9:0]	Microengine PC
W1[0]	UEngDbgValid	ue_xxx_DbgValid_c2a	Microengine Debug Valid Flag [See Note 1]

Note 1: W1[0] = ue_xxx_DbgValid_c2a was a mistake, asserts 2 cycles before the other Dbg signals, it should have been ue_xxx_DbgValid_c4a. But since ue_xxx_DbgValid_c4a is available as one of the triggers, you can still achieve qualification by DbgValid by just including W0[14] (ue_xxx_DbgValid_c4a) == 1 as part of the equation for a match.

11.14.1.3 DMA Vector Trigger Inputs (Mux 1)

DMA Engine transmit and receive port reference counts.

Class

TrbvDmaMux1

Attributes

-ocla -trbv -trbvdma

Bit	Mnemonic	(Signal)	Definition
W0[31:29]			Reserved
W0[28:25]	CswToDmaCmdOrigin	csw_dma_CmdOrigin_c1a	Origin of CSW command
W0[24]	CifToRxp0RefCntZero	cif_rxp0_RefCntZero_c5a	Receive port 0 reference count is zero
W0[23]	CifToRxp1RefCntZero	cif_rxp1_RefCntZero_c5a	Receive port 1 reference count is zero
W0[22]	CifToRxp2RefCntZero	cif_rxp2_RefCntZero_c5a	Receive port 2 reference count is zero
W0[21]	CifRxRefCntZero	cif_copy_RxRefCntZero_c5a	Copy receive reference count is zero
W0[20]	CifToTxp0RefCntZero	cif_txp0_RefCntZero_c5a	Transmit port 0 reference count is zero
W0[19]	CifToTxp1RefCntZero	cif_txp1_RefCntZero_c5a	Transmit port 1 reference count is zero
W0[18]	CifToTxp2RefCntZero	cif_txp2_RefCntZero_c5a	Transmit port 2 reference count is zero
W0[17]	CifTxRefCntZero	cif_copy_TxRefCntZero_c5a	Copy transmit reference count is zero
W0[16]	CifToUEngStartIo	cif_ue_StartIo_c1a	Microengine IO Start
W0[15:14]	CifToUEngStartIoType	cif_ue_StartIoType_c1a[1:0]	Microengine IO Start Type
W0[13:10]	CifToUEngStartIoAddr	cif_ue_StartIoAddr_c1a[6:3]	Microengine IO Start Address
W0[9]	UEngToCifRdyForStartIo	ue_cif_RdyForStartIo_c3a	Microengine ready for Start IO
W0[8]	UEngToCifTaskStart	ue_cif_TaskStart_c5a	Microengine task start
W0[7:4]	UEngToCifTaskThread	ue_cif_TaskThread_c5a[3:0]	Microengine task thread
W0[3:0]	UEngToCifTaskType	ue_cif_TaskType_c5a[3:0]	Microengine task type
W1[0]	UEngDbgValid	ue_xxx_DbgValid_c2a	Microengine Debug Valid Flag

11.14.1.4 DMA Vector Trigger Inputs (Mux 2)

DMA Engine's central switch to transmit/receive port interfaces.

Class

TrbvDmaMux2

Attributes

-ocla -trbv -trbvdma

Bit	Mnemonic	(Signal)	Definition
W0[31:23]			Reserved
W0[22]	CifMemInPbufSel	cif_copy_MemInPbufSel_c4a	tbs
W0[21]	CifMemInRmbSel	cif_copy_MemInRmbSel_c4a	tbs
W0[20]	CifToTxpMemInTxp0Sel	cif_txp_MemInTxp0Sel_c4a	tbs
W0[19]	CifToTxpMemInTxp1Sel	cif_txp_MemInTxp1Sel_c4a	tbs
W0[18]	CifToTxpMemInTxp2Sel	cif_txp_MemInTxp2Sel_c4a	tbs
W0[17]	CifMemOutPbufSel	cif_copy_MemOutPbufSel_c2a	tbs
W0[16]	CifMemOutWmbSel	cif_copy_MemOutWmbSel_c2a	tbs
W0[15]	CifToRxpMemOutRxp0Sel	cif_rxp_MemOutRxp0Sel_c2a	tbs
W0[14]	CifToRxpMemOutRxp1Sel	cif_rxp_MemOutRxp1Sel_c2a	tbs
W0[13]	CifToRxpMemOutRxp2Sel	cif_rxp_MemOutRxp2Sel_c2a	tbs
W0[12]	CifToRxpMemOutCopySel	cif_rxp_MemOutCopySel_c2a	tbs
W0[11:8]	CifMemInAlign	cif_xxx_MemInAlign_c4a[3:0]	tbs
W0[7:0]	CifMemInAddr	cif_xxx_MemInAddr_c4a[7:0]	tbs
W1[0]	UEngDbgValid	ue_xxx_DbgValid_c2a	Microengine Debug Valid Flag

11.14.1.5 DMA Vector Trigger Inputs (Mux 3)

DMA Engine internal memory writes.

Class

TrbvDmaMux3

Attributes

-ocla -trbv -trbvdma

Bit	Mnemonic	(Signal)	Definition
W0[31]	DmemResultSel	ue_dmem_ResultSel_c5a	Asserted when dmem is written by an instruction
W0[30:21]	DmemResultAddr	ue_xxx_ResultAddr_c5a	Address in dmem where ALU result is written
W0[20:0]	DmemResultData	alu_xxx_ResultDat_c5a[20:0]	ALU result to be written to dmem
W1[0]	UEngDbgValid	ue_xxx_DbgValid_c2a	Microengine Debug Valid Flag

11.14.2 DMA Collector

The DMA engine has a single 1024 x 33 bit CTB. Its inputs are configured identically to those for the vector TRB in the DMA engine. (See Tables 11.14.1.2, 11.14.1.3, 11.14.1.4 and 11.14.1.5.)

11.14.2.1 DMA Input Collectors Qualifying Triggers

The CTB has two qualifier inputs. Qtrig[1] is connected to ue_xxx_DbgValid_c2a, and Qtrig[0] is connected to ue_cif_TaskStart_c5a.

Class

CtbDmaQtrig

Attributes

-ocla -ctb -ctbdma

Bit	Mnemonic	(Signal)	Definition
1	UEngDbgValid	dma.csr.ue_xxx_DbgValid_c2a	Microengine Debug Valid Flag [Broken, see Note 1]
0	UEngToCifTaskStart	dma.csr.ue_cif_TaskStart_c5a	Microengine To CIF Task Start

Note 1:

This is broken, it should have been connected to ue_xxx_DbgValid_c4a in order to allow us to collapse collection of “Dbg” signals. With it connected to ue_xxx_DbgValid_c2a we effectively cannot use this collection qualifier at all.

11.14.2.2 DMA Input Collector Mux 0

Class

CtbDmaMux0

Attributes

-ocla -ctb -ctbdma

Bit	Mnemonic	(Signal)	Definition
31	Rxp0ToUEngBufAvail	dma.rxp0_ue_BufAvail_c1a	Receive port 0 to microengine buffer available
30	Rxp1ToUEngBufAvail	dma.rxp1_ue_BufAvail_c1a	Receive port 1 to microengine buffer available
29	Rxp2ToUEngBufAvail	dma.rxp2_ue_BufAvail_c1a	Receive port 2 to microengine buffer available
28	UEngRxThreadStart	dma.copy_ue_RxThreadStart_c1a	Microengine receive thread start
27	Txp0ToUEngBufAvail	dma.txp0_ue_BufAvail_c1a	Transmit port 0 to microengine buffer available
26	Txp1ToUEngBufAvail	dma.txp1_ue_BufAvail_c1a	Transmit port 1 to microengine buffer available
25	Txp2ToUEngBufAvail	dma.txp2_ue_BufAvail_c1a	Transmit port 2 to microengine buffer available
24	UEngTxThreadStart	dma.copy_ue_TxThreadStart_c1a	Microengine transmit thread start
23	UEngToRxp0BufXfr	dma.ue_rxp0_BufTransfer_c5a	Microengine to receive port 0 buffer transfer
22	UEngToRxp1BufXfr	dma.ue_rxp1_BufTransfer_c5a	Microengine to receive port 1 buffer transfer
21	UEngToRxp2BugXfr	dma.ue_rxp2_BufTransfer_c5a	Microengine to receive port 2 buffer transfer
20	UEngRxThreadDone	dma.ue_copy_RxThreadDone_c5a	Microengine receive thread done
19	UEngToTxp0BufXfr	dma.ue_txp0_BufTransfer_c5a	Microengine to transmit port 0 buffer transfer
18	UEngToTxp1BufXfr	dma.ue_txp1_BufTransfer_c5a	Microengine to transmit port 1 buffer transfer
17	UEngToTxp2BufXfr	dma.ue_txp2_BufTransfer_c5a	Microengine to transmit port 2 buffer transfer
16	UEngTxThreadDone	dma.ue_copy_TxThreadDone_c5a	Microengine transmit thread done
15		Unused	Reserved
14	UEngDbgValid	dma.csr.m_ue_xxx_DbgValid_c4a	Microengine thread valid
13:10	UEngDbgThread	dma.csr.m_ue_xxx_DbgThread_c4a[3:0]	Microengine thread number
9:0	UEngDbgPc	dma.ue_xxx_DbgPc_c4a[9:0]	Microengine PC

11.14.2.3 DMA Input Collector Mux 1

Class

CtbDmaMux1

Attributes

-ocla -ctb -ctbdma

Bit	Mnemonic	(Signal)	Definition
31:29			Reserved
28:25	CswToDmaCmdOrigin	csw_dma_CmdOrigin_c1a	Origin of CSW command
24	CifToRxp0RefCntZero	cif_rxp0_RefCntZero_c5a	Receive port 0 reference count is zero
23	CifToRxp1RefCntZero	cif_rxp1_RefCntZero_c5a	Receive port 1 reference count is zero
22	CifToRxp2RefCntZero	cif_rxp2_RefCntZero_c5a	Receive port 2 reference count is zero
21	CifRxRefCntZero	cif_copy_RxRefCntZero_c5a	Copy receive reference count is zero
20	CifToTxp0RefCntZero	cif_txp0_RefCntZero_c5a	Transmit port 0 reference count is zero
19	CifToTxp1RefCntZero	cif_txp1_RefCntZero_c5a	Transmit port 1 reference count is zero
18	CifToTxp2RefCntZero	cif_txp2_RefCntZero_c5a	Transmit port 2 reference count is zero
17	CifTxRefCntZero	cif_copy_TxRefCntZero_c5a	Copy transmit reference count is zero
16	CifToUEngStartIo	cif_ue_StartIo_c1a	Microengine IO Start
15:14	CifToUEngStartIoType	cif_ue_StartIoType_c1a[1:0]	Microengine IO Start Type
13:10	CifToUEngStartIoAddr	cif_ue_StartIoAddr_c1a[6:3]	Microengine IO Start Address
9	UEngToCifRdyForStartIo	ue_cif_RdyForStartIo_c3a	Microengine ready for Start IO
8	UEngToCifTaskStart	ue_cif_TaskStart_c5a	Microengine task start
7:4	UEngToCifTaskThread	ue_cif_TaskThread_c5a[3:0]	Microengine task thread
3:0	UEngToCifTaskType	ue_cif_TaskType_c5a[3:0]	Microengine task type

11.14.2.4 DMA Input Collector Mux 2

Class

CtbDmaMux2

Attributes

-ocla -ctb -ctbdma

Bit	Mnemonic	(Signal)	Definition
31:23			Reserved
22	CifMemInPbufSel	cif_copy_MemInPbufSel_c4a	tbs
21	CifMemInRmbSel	cif_copy_MemInRmbSel_c4a	tbs
20	CifToTxpMemInTxp0Sel	cif_txp_MemInTxp0Sel_c4a	tbs
19	CifToTxpMemInTxp1Sel	cif_txp_MemInTxp1Sel_c4a	tbs
18	CifToTxpMemInTxp2Sel	cif_txp_MemInTxp2Sel_c4a	tbs
17	CifMemOutPbufSel	cif_copy_MemOutPbufSel_c2a	tbs
16	CifMemOutWmbSel	cif_copy_MemOutWmbSel_c2a	tbs
15	CifToRxpMemOutRxp0Sel	cif_rxp_MemOutRxp0Sel_c2a	tbs
14	CifToRxpMemOutRxp1Sel	cif_rxp_MemOutRxp1Sel_c2a	tbs
13	CifToRxpMemOutRxp2Sel	cif_rxp_MemOutRxp2Sel_c2a	tbs
12	CifToRxpMemOutCopySel	cif_rxp_MemOutCopySel_c2a	tbs
11:8	CifMemInAlign	cif_xxx_MemInAlign_c4a[3:0]	tbs
7:0	CifMemInAddr	cif_xxx_MemInAddr_c4a[7:0]	tbs

11.14.2.5 DMA Input Collector Mux 3

Class

CtbDmaMux3

Attributes

-ocla -ctb -ctbdma

Bit	Mnemonic	(Signal)	Definition
31	DmemResultSel	ue_dmem_ResultSel_c5a	Asserted when dmem is written by an instruction
30:21	DmemResultAddr	ue_xxx_ResultAddr_c5a	Address in dmem where ALU result is written
20:0	DmemResultData	alu_xxx_ResultDat_c5a[20:0]	ALU result to be written to dmem

11.14.2.6 DMA Input Collector Mux 4, 5, 6, 7

Collects all-zeros.

11.15 OCLA in use – PMI

11.15.1 PMI/PCI/BBS Triggers

The PMI/PCI/BBS contains two codeword trigger units. The first trigger unit is on its CSW bus stop and the second trigger unit is for signals internal to the PMI.

11.15.1.1 "TrbcPmi" PMI CSW Bus Stop Codeword Triggers

The CSW side of the PMI is connected to the first TRBC unit with connections as shown below. This "TrbcPmi" is "trbc0" in the Verilog source code file PmiOcl.v. Note that for all TRBs, word x (that is W0, W1, W2, W3) maps to CodeSampX (CodeSamp0, CodeSamp1... respectively.) W4 and W5 map to the two CodeValid inputs.

No external mux is used, there is only one set of signals wired to this trigger block. Field "ExtMuxSel" of R_TrbcPmiTrigCtl has no effect, can be left unchanged, or written to any value.

Class

TrbcPmi

Attributes

-ocla -trbc -trbcpmi

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4:0]	CswToPmiCommand	xxx_trbc_CodeSamp0_x0a[4:0]	csw_pmi_Command_c1a	Inbound Command Code from CSW
W1[4:0]	CswToPmiCmdAddrTID	xxx_trbc_CodeSamp1_x0a[4:0]	csw_pmi_CmdAddrTID_c1a	Inbound Request Transaction ID from CSW
W2[4:0]	CswToPmiDataTID	xxx_trbc_CodeSamp2_x0a[4:0]	csw_pmi_DataTID_c3a	Inbound Data Transaction ID from CSW
W3[4]	PmiToCswECmdAddReq	xxx_trbc_CodeSamp3_x0a[4]	pmi_csw_ECmdAddrReq_c0a	Outbound to COHE Command Request from PCI
W3[3]	PmiToCswOCmdAddrReq	xxx_trbc_CodeSamp3_x0a[3]	pmi_csw_OCmdAddrReq_c0a	Outbound to COHO Command Request from PCI
W3[2]	CswToPmiCmdAddrGnt	xxx_trbc_CodeSamp3_x0a[2]	csw_pmi_CmdAddrGnt_c1a	Inbound Command Grant to PCI
W3[1]		xxx_trbc_CodeSamp3_x0a[1]		Reserved (Always '0')
W3[0]		xxx_trbc_CodeSamp3_x0a[0]		Reserved (Always '0')
W4[0]	Cv0CswToPmiCmdAddrValid	xxx_trbc_CodeValid0_x0a	csw_pmi_CmdAddrValid_c1a	Command/Transfer Valid, CSW is sending cmd to PCI

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W5[0]	Cv1CswToPmiDataValid	xxx_trbc_CodeValid1_x0a	csw_pmi_DataValid_c3a	CSW is sending data to PCI

11.15.1.2 "TrbcPmii" PMI Internal Signal Codeword Triggers

The following PMI internal signals are connected to the second TRBC unit as shown below. This "TrbcPmii" is "trbc1" in the Verilog source code file PmiOcl.v.

No external mux is used, there is only one set of signals wired to this trigger block.

Due to Bug 1959, affecting PMI only in Ice9A, the ExtMuxSel field of R_TrbcPmiiTrigCtl is the mux-select of input signals for PMI's CTB! This is fixed in Ice9B.

The value 7 has no special "power-savings" meaning like in other units. In PMI it selects a set of signals to collect. Field ExtMuxSel in R_CtbPmiColCtl does nothing, can be left unchanged or set to any value.

Class

TrbcPmii

Attributes

-ocla -trbc -trbcpmi

Bit	Mnemonic	(Codeword Sample Input)	(Signal)	Definition
W0[4:2]		xxx_trbc_CodeSamp0_x0a[4:2]		Reserved
W0[1]	SycToCcrRdHdrValid	xxx_trbc_CodeSamp0_x0a[1]	m_RdHdrVal_c1a	Flopped syc_ccr_RdHdrVal_c0a valid bit for header
W0[0]	CmdInProcess	xxx_trbc_CodeSamp0_x0a[0]	m_CommandInProcess_c1a	A command is being processed
W1[4]	SycToCcwWrHdrValid	xxx_trbc_CodeSamp1_x0a[4]	syc_ccw_WrHdrVal_c0a	tbs (flopped one more time than m_WrSmState_c1a)
W1[3:0]	WrSmState	xxx_trbc_CodeSamp1_x0a[3:0]	m_WrSmState_c1a[3:0]	tbs
W2[4]	RrfToCcmSetValid	xxx_trbc_CodeSamp2_x0a[4]	rrf_ccm_SetValid_c4a	tbs
W2[3]	CxdToRrfCmdValid	xxx_trbc_CodeSamp2_x0a[3]	cxd_rrf_CmdValid_c4a	tbs
W2[2]	CcrToSycRdHdrPop	xxx_trbc_CodeSamp2_x0a[2]	ccr_syc_RdHdrPop_c1a	tbs
W2[1]	CcwToSycDatPop	xxx_trbc_CodeSamp2_x0a[1]	ccw_syc_DatPop_c1a	tbs
W2[0]	CcwToSycWrHdrPop	xxx_trbc_CodeSamp2_x0a[0]	ccw_syc_WrHdrPop_c1a	tbs
W3[4]	UartToPmiWishbAck	xxx_trbc_CodeSamp3_x0a[4]	uart_pmi_wbAck_c	Wishbone ack from UART core.
W3[3]	PmiToI2cWishbStrobe	xxx_trbc_CodeSamp3_x0a[3]	pmi_i2c_wbStrobe_c	Wishbone strobe to I2C core.
W3[2]	PmiToUartWishbStrobe	xxx_trbc_CodeSamp3_x0a[2]	pmi_uart_wbStrobe_c	Wishbone strobe to UART core.
W3[1]	I2cToPmiWishbAck	xxx_trbc_CodeSamp3_x0a[1]	i2c_pmi_wbAck_c	Wishbone ack from I2C core.
W3[0]	PmiWishbCycle	xxx_trbc_CodeSamp3_x0a[0]	pmi_ui2c_wbCycle_c	Wishbone cycle signal from PMI.
W4[0]	Cv0Always1	xxx_trbc_CodeValid0_x0a[0]		Always '1'
W5[0]	Cv1Always1	xxx_trbc_CodeValid1_x0a[0]		Always '1'

11.15.2 PMI/PCI/BBS Collector

The PMI/PCI/BBS contains one 1024 x 33 bit CTB, with an external mux to select sets of signals to collect.

Due to Bug 1959, affecting PMI only in Ice9A, the ExtMuxSel field of R_TrbcPmiiTrigCtl is the mux-select of input signals for PMI's CTB! This is fixed in Ice9B.

The value 7 has no special "power-savings" meaning like in other units. In PMI it selects a set of signals to collect. Field ExtMuxSel in R_CtbPmiColCtl does nothing, can be left unchanged or set to any value.

11.15.2.1 PMI Input Qualifying Triggers

Class

CtbPmiQtrig

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
1	Qtrig1Always1	xxx_ctb_QualTrigger1_x0a	m_high	Always at '1'
0	Qtrig0Always1	xxx_ctb_QualTrigger0_x0a	m_high	Always at '1'

11.15.2.2 PMI Input Collector Mux 0

Class

CtbPmiMux0

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:29		xxx_ctb_SampleDataIn0_x0a[31:29]		Reserved
28	SycToCcrRdHdrVal	xxx_ctb_SampleDataIn0_x0a[28]	m_RdHdrVal_c1a	Flopped syc_ccr_RdHdrVal_c0a, valid bit for header
27:24	SycToCcrRdLastBe0	xxx_ctb_SampleDataIn0_x0a[27:24]	m_RdLastBe_c1a[3:0]	Flopped syc_ccr_RdLastBe_c0a[3:0]
23:20	SycToCcrRdFirstBe0	xxx_ctb_SampleDataIn0_x0a[23:20]	m_RdFirstBe_c1a[3:0]	Flopped syc_ccr_RdFirstBe_c0a[3:0]
19:10	SycToCcrRdDwLen0	xxx_ctb_SampleDataIn0_x0a[19:10]	m_RdDwLen_c1a[9:0]	Flopped syc_ccr_RdDwLen_c0a[9:0] lower 10 bits of 11.
9	CcrToSycRdHdrPop0	xxx_ctb_SampleDataIn0_x0a[9]	m_CcrSycRdHdrPop_c2a	Flopped ccr_syc_RdHdrPop_c1a
8	Buf2Busy	xxx_ctb_SampleDataIn0_x0a[8]	m_Buf2Busy_c6a	tbs
7	Buf1Busy	xxx_ctb_SampleDataIn0_x0a[7]	m_Buf1Busy_c6a	tbs
6	Buf0Busy	xxx_ctb_SampleDataIn0_x0a[6]	m_Buf0Busy_c6a	tbs
5	Serv2	xxx_ctb_SampleDataIn0_x0a[5]	m_Servicing2_c7a	tbs
4	Serv1	xxx_ctb_SampleDataIn0_x0a[4]	m_Servicing1_c7a	tbs
3	Serv0	xxx_ctb_SampleDataIn0_x0a[3]	m_Servicing0_c7a	tbs
2	CmdInProgress	xxx_ctb_SampleDataIn0_x0a[2]	m_CommandInProgress_c1a	tbs
1:0	RdSmState	xxx_ctb_SampleDataIn0_x0a[1:0]	m_RdSmState_c1a[1:0]	tbs

11.15.2.3 PMI Input Collector Mux 1

Class

CtbPmiMux1

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31	SycToCcwWrHdrVal1	xxx_ctb_SampleDataIn1_x0a[31]	syc_ccw_WrHdrVal_c0a	Write Header Valid
30:27	SycToCcwWrLastBe1	xxx_ctb_SampleDataIn1_x0a[30:27]	syc_ccw_WrLastBe_c0a[3:0]	tbs
26:23	SycToCcwWrFirstBe1	xxx_ctb_SampleDataIn1_x0a[26:23]	syc_ccw_WrFirstBe_c0a[3:0]	tbs
22:13	SycToCcwWrDwLen1	xxx_ctb_SampleDataIn1_x0a[22:13]	syc_ccw_WrDwLen_c0a[9:0]	tbs, the lowest 10 bits of 11-bit WrDwLen
12	CcwToSycWrHdrPop1	xxx_ctb_SampleDataIn1_x0a[12]	ccw_syc_WrHdrPop_c1a	tbs
11:5	CcwWrSeqNum	xxx_ctb_SampleDataIn1_x0a[11:5]	ccw_xxx_WrSeqNum_c1a[6:0]	tbs, the lowest 7 bits of 11-bit WrSeqNum
4	CmdBusy	xxx_ctb_SampleDataIn1_x0a[4]	m_CmdBusy_c2a	tbs (flopped one less time than signals above)
3:0	WrSmState	xxx_ctb_SampleDataIn1_x0a[3:0]	m_WrSmState_c1a[3:0]	tbs (flopped one less time than signals above)

11.15.2.4 PMI Input Collector Mux 2

Class

CtbPmiMux2

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:30		xxx_ctb_SampleDataIn2_x0a[31:30]		Reserved
29	CcwToSycDatPop	xxx_ctb_SampleDataIn2_x0a[29]	ccw_syc_DatPop_c1a	tbs
28	CcwToSycWrHdrPop2	xxx_ctb_SampleDataIn2_x0a[28]	ccw_syc_WrHdrPop_c1a	tbs
27	SycToCcwWrHdrVal2	xxx_ctb_SampleDataIn2_x0a[27]	syc_ccw_WrHdrVal_c0a	tbs
26:23	SycToCcwWrLastBe2	xxx_ctb_SampleDataIn2_x0a[26:23]	syc_ccw_WrLastBe_c0a[3:0]	tbs
22:19	SycToCcwWrFirstBe2	xxx_ctb_SampleDataIn2_x0a[22:19]	syc_ccw_WrFirstBe_c0a[3:0]	tbs
18:9	SycToCcwWrDwLen2	xxx_ctb_SampleDataIn2_x0a[18:9]	syc_ccw_WrDwLen_c0a[9:0]	tbs, the lowest 10 bits of 11-bit WrDwLen
8		xxx_ctb_SampleDataIn2_x0a[8]		Reserved
7:0	SycToCcwWrReqTag	xxx_ctb_SampleDataIn2_x0a[7:0]	syc_ccw_WrReqTag_c0a[7:0]	tbs

11.15.2.5 PMI Input Collector Mux 3

Class

CtbPmiMux3

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:28		xxx_ctb_SampleDataIn3_x0a[31:28]		Reserved
27	CcrToSycRdHdrPop3	xxx_ctb_SampleDataIn3_x0a[27]	ccr_syc_RdHdrPop_c1a	tbs
26	SycToCcrRdHalt	xxx_ctb_SampleDataIn3_x0a[26]	syc_ccr_RdHalt_c0a	tbs
25:19	SycToCcrRdSeqNum	xxx_ctb_SampleDataIn3_x0a[25:19]	syc_ccr_RdSeqNum_c0a[6:0]	tbs, the lowest 7 bits of 11-bit RdSeqNum
18	SycToCcrRdHdrVal3	xxx_ctb_SampleDataIn3_x0a[18]	syc_ccr_RdHdrVal_c0a	tbs
17:14	SycToCcrRdLastBe3	xxx_ctb_SampleDataIn3_x0a[17:14]	syc_ccr_RdLastBe_c0a[3:0]	tbs
13:10	SycToCcrRdFirstBe3	xxx_ctb_SampleDataIn3_x0a[13:10]	syc_ccr_RdFirstBe_c0a[3:0]	tbs
9:0	SycToCcrRdDwLen3	xxx_ctb_SampleDataIn3_x0a[9:0]	syc_ccr_RdDwLen_c0a[9:0]	tbs, lower 10 bits of 11.

11.15.2.6 PMI Input Collector Mux 4

Class

CtbPmiMux4

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:29		xxx_ctb_SampleDataIn4_x0a[31:29]		Reserved
28	CcmToCxdDone	xxx_ctb_SampleDataIn4_x0a[28]	ccm_cxd_Done_c3a	tbs
27	RrfToCxdCmdEmpty	xxx_ctb_SampleDataIn4_x0a[27]	rrf_cxd_CmdEmpty_c3a	tbs
26	CxdToRrfCmdValid	xxx_ctb_SampleDataIn4_x0a[26]	cxd_rrf_CmdValid_c4a	tbs
25:21	CxdToRrfTid	xxx_ctb_SampleDataIn4_x0a[25:21]	cxd_rrf_TID_c4a[4:0]	tbs
20:13	CxdToRrfBMask	xxx_ctb_SampleDataIn4_x0a[20:13]	cxd_rrf_BMask_c4a[7:0]	tbs

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
12:8	CxdToRrfCmd	xxx_ctb_SampleDataIn4_x0a[12:8]	cxd_rrf_Cmd_c4a[4:0]	tbs
7:4	CxdToRrfCmdOrigin	xxx_ctb_SampleDataIn4_x0a[7:4]	cxd_rrf_CmdOrigin_c4a[3:0]	tbs
3	WriteOutstanding	xxx_ctb_SampleDataIn4_x0a[3]	m_WriteOutstanding_c4a	tbs
2	DataNeeded	xxx_ctb_SampleDataIn4_x0a[2]	m_DataNeeded_c4a	tbs
1	DataValid	xxx_ctb_SampleDataIn4_x0a[1]	m_DataValid_c4a	tbs
0	NpOpWait	xxx_ctb_SampleDataIn4_x0a[0]	m_NpOpWait_c4a	tbs

11.15.2.7 PMI Input Collector Mux 5

Class

CtbPmiMux5

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:0		xxx_ctb_SampleDataIn5_x0a[31:0]		Reserved, all zeros

11.15.2.8 PMI Input Collector Mux 6

Class

CtbPmiMux6

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:25	CcwWrSeqNum	xxx_ctb_SampleDataIn6_x0a[31:25]	ccw_xxx_WrSeqNum_c1a[6:0]	tbs
24:18	RrfToCcmSeqNum	xxx_ctb_SampleDataIn6_x0a[24:18]	rrf_ccm_SeqNum_c4a[6:0]	tbs
17:15	RrfToCcmCplState	xxx_ctb_SampleDataIn6_x0a[17:15]	rrf_ccm_CplState_c4a[2:0]	tbs
14	RrfToCcmSetValid	xxx_ctb_SampleDataIn6_x0a[14]	rrf_ccm_SetValid_c4a	tbs
13	CcmToCxdDone	xxx_ctb_SampleDataIn6_x0a[13]	ccm_cxd_Done_c3a	tbs
12	CxdToCcmCmdVal	xxx_ctb_SampleDataIn6_x0a[12]	cxd_ccm_CmdVal	tbs
11	CxdToCcmLinkDwn	xxx_ctb_SampleDataIn6_x0a[11]	cxd_ccm_LinkDwn_c4a	tbs
10	CxdToCcmRdOp	xxx_ctb_SampleDataIn6_x0a[10]	cxd_ccm_RdOp_c4a	tbs
9		xxx_ctb_SampleDataIn6_x0a[9]		Reserved, always 0 (See Note 1)
8:5	CxdToCcmDest	xxx_ctb_SampleDataIn6_x0a[8:5]	cxd_ccm_Dest_c4a[3:0]	tbs
4:0	CxdToCcmTid	xxx_ctb_SampleDataIn6_x0a[4:0]	cxd_ccm_TID_c4a[4:0]	tbs

Note 1:
In the verilog RTL, cxd_ccm_Dest_c4a is [3:0], but in the behavioral model it's [4:0]. In the behavioral model xxx_ctb_SampleDataIn6_x0a[9] is connected to cxd_ccm_Dest_c4a[4], although it should always simulate with this bit = 0.

11.15.2.9 PMI Input Collector Mux 7

Class

CtbPmiMux7

Attributes

-ocla -ctb -ctbpmi

Bit	Mnemonic	(CTB Input)	(Signal)	Definition
31:24	PmiWishbOutData	xxx_ctb_SampleDataIn7_x0a[31:24]	pmi_ui2c_wbDatO_c[7:0]	Outbound PMI Wishbone Data
23:16	PmiWishbInData	xxx_ctb_SampleDataIn7_x0a[23:16]	ml_WbDti_c5a[7:0]	Muxed Inbound PMI Wishbone Data
15:9		xxx_ctb_SampleDataIn7_x0a[15:9]		Reserved
8	UartToPmiWishbAck	xxx_ctb_SampleDataIn7_x0a[8]	uart_pmi_wbAck_c	UART to PMI Wishbone Ack
7	PmiToI2cWishbStrobe	xxx_ctb_SampleDataIn7_x0a[7]	pmi_i2c_wbStrobe_c	PMI to I2C Wishbone Strobe
6	PmiToUartWishbStrobe	xxx_ctb_SampleDataIn7_x0a[6]	pmi_uart_wbStrobe_c	PMI to UART Wishbone Strobe
5	I2cToPmiWishbAck	xxx_ctb_SampleDataIn7_x0a[5]	i2c_pmi_wbAck_c	I2C to PMI Wishbone Ack
4	PmiWishbCycle	xxx_ctb_SampleDataIn7_x0a[4]	pmi_ui2c_wbCycle_c	Outbound PMI Wishbone Cycle
3	PmiWishbWriteEnb	xxx_ctb_SampleDataIn7_x0a[3]	pmi_ui2c_wbWriteEnb_c	Outbound PMI Wishbone Write Enable
2:0	PmiWishbAddr	xxx_ctb_SampleDataIn7_x0a[2:0]	pmi_ui2c_wbAdr_c[2:0]	Outbound PMI Wishbone Address

11.16 Register Address Ranges

11.16.1 TrbcPs0

Register

R_TrbcPs0* : R_Trbcx*

Address

0xE_0C00_0000-0xE_0CFF_FFFF

11.16.2 TrbcPs1

Register

R_TrbcPs1* : R_Trbcx*

Address

0xE_1C00_0000-0xE_1CFF_FFFF

11.16.3 TrbcPs2

Register

R_TrbcPs2* : R_Trbcx*

Address

0xE_2C00_0000-0xE_2CFF_FFFF

11.16.4 TrbcPs3

Register

R_TrbcPs3* : R_Trbcx*

Address

0xE_3C00_0000-0xE_3CFF_FFFF

11.16.5 TrbcPs4

Register

R_TrbcPs4* : R_Trbcx*

Address

0xE_4C00_0000-0xE_4CFF_FFFF

11.16.6 TrbcPs5

Register

R_TrbcPs5* : R_Trbcx*

Address

0xE_5C00_0000-0xE_5CFF_FFFF

11.16.7 TrbcPs6

Register

R_TrbcPs6* : R_Trbcx*

Attributes

-Product=TWC9A+

Address

0xE_4900_0000-0xE_49FF_FFFF

11.16.8 TrbcPs7

Register

R_TrbcPs7* : R_Trbcx*

Attributes

-Product=TWC9A+

Address

0xE_5900_0000-0xE_59FF_FFFF

11.16.9 TrbcPs8

Register

R_TrbcPs8* : R_Trbcx*

Attributes

-Product=TWC9A+

Address

0xE_6900_0000-0xE_69FF_FFFF

11.16.10 TrbcPs9

Register

R_TrbcPs9* : R_Trbcx*

Attributes

-Product=TWC9A+

Address

0xE_7900_0000-0xE_79FF_FFFF

11.16.11 TrbcDma

Register

R_TrbcDma* : R_Trbcx*

Address

0xE_6C00_0000-0xE_6CFF_FFFF

11.16.12 TrbvDma

Register

R_TrbvDma* : R_Trbvx*

Address

0xE_7C00_0000-0xE_7CFF_FFFF

11.16.13 TrbcPmi

Register

R_TrbcPmi* : R_Trbcx*

Address

0xE_0F00_0000-0xE_0FFF_FFFF

11.16.14 TrbcPmii

Register

R_TrbcPmii* : R_Trbcx*

Address

0xE_4F00_0000-0xE_4FFF_FFFF

11.16.15 TrbcCoho

Register

R_TrbcCoho* : R_Trbcx*

Address

0xE_3A00_0000-0xE_3AFF_FFFF

11.16.16 TrbcCohe

Register

R_TrbcCohe* : R_Trbcx*

Address

0xE_2A00_0000-0xE_2AFF_FFFF

11.16.17 TrbvFswo

Register

R_TrbvFswo* : R_Trbvx*

Address

0xE_1F00_0000-0xE_1FFF_FFFF

11.16.18 TrbvFswi

Register

R_TrbvFswi* : R_Trbvx*

Address

0xE_2F00_0000-0xE_2FFF_FFFF

11.16.19 TrbcFsw

Register

R_TrbcFsw* : R_Trbcx*

Address

0xE_3F00_0000-0xE_3FFF_FFFF

11.16.20 CtbPs0

Register

R_CtbPs0* : R_Ctbx*

Address

0xE_0B00_0000-0xE_0BFF_FFFF

11.16.21 CtbPs1

Register

R_CtbPs1* : R_Ctbx*

Address

0xE_1B00_0000-0xE_1BFF_FFFF

11.16.22 CtbPs2

Register

R_CtbPs2* : R_Ctbx*

Address

0xE_2B00_0000-0xE_2BFF_FFFF

11.16.23 CtbPs3

Register

R_CtbPs3* : R_Ctbx*

Address

0xE_3B00_0000-0xE_3BFF_FFFF

11.16.24 CtbPs4

Register

R_CtbPs4* : R_Ctbx*

Address

0xE_4B00_0000-0xE_4BFF_FFFF

11.16.25 CtbPs5

Register

R_CtbPs5* : R_Ctbx*

Address

0xE_5B00_0000-0xE_5BFF_FFFF

11.16.26 CtbPs6

Register

R_CtbPs6* : R_Ctbx*

Attributes

-Product=TWC9A+

Address

0xE_4100_0000-0xE_41FF_FFFF

11.16.27 CtbPs7

Register

R_CtbPs7* : R_Ctbx*

Attributes

-Product=TWC9A+

Address

0xE_5100_0000-0xE_51FF_FFFF

11.16.28 CtbPs8

Register

R_CtbPs8* : R_Ctbx*

Attributes

-Product=TWC9A+

Address

0xE_6100_0000-0xE_61FF_FFFF

11.16.29 CtbPs9

Register

R_CtbPs9* : R_Ctbx*

Attributes

-Product=TWC9A+

Address

0xE_7100_0000-0xE_71FF_FFFF

11.16.30 CtbDma

Register

R_CtbDma* : R_Ctbx*

Address

0xE_6B00_0000-0xE_6BFF_FFFF

11.16.31 CtbPmi

Register

R_CtbPmi* : R_Ctbx*

Address

0xE_7B00_0000-0xE_7BFF_FFFF

11.16.32 CtbCoho

Register

R_CtbCoho* : R_Ctbx*

Address

0xE_1A00_0000-0xE_1AFF_FFFF

11.16.33 CtbCohe

Register

R_CtbCohe* : R_Ctbx*

Address

0xE_0A00_0000-0xE_0AFF_FFFF

11.16.34 CtbFswi

Register

R_CtbFswi* : R_Ctbx*

Address

0xE_4A00_0000-0xE_4AFF_FFFF

11.16.35 CtbFswo

Register

R_CtbFswo* : R_Ctbx*

Address

0xE_5A00_0000-0xE_5AFF_FFFF

11.17 OCLA Programming Suggestions

11.17.1 Ready-To-Use OCLA Scripts

Available scripts for using OCLA are documented in: **<project>/specs/diags/DiagnosticOCLA.lyx**
Some pre-written OCLA scripts for the diagnostics "dash" environment are in: **<project>/diags/ocla_test/**
These allow you to use OCLA with a few short commands in simple cases where a per-unit trigger is not needed.
For easy diagnostics dash control of OCLA, whether using the above-mentioned scripts, or your own configuration, look in: **<project>/diags/ocla/**

11.17.2 Example Code for OCLA

For examples of OCLA programming, look at the simulation tests we wrote to verify OCLA.
Most of the OCLA simulation tests are listed and described on Wiki page: **http://apollo.sicortex.com/swiki/OclaVerific**
Commands to simulate these tests are (under svn rev control) in: **<project>/hw/tests/testlists/ocla_use.vtest**
Source code (under svn rev control) is in directory: **<project>/sw/anthrax/tests/ocla/**
Each overall OCLA "program" in this directory requires 2 files and has 3 major parts. For example, test "ocla_ps3_t1c2q_biuwr" is coded in files ocla_ps3_t1c2q_biuwr.c and ocla_ps3_t1c2q_biuwr_util.cpp. The _util.cpp file contains 2 parts, the upper part creates the OCLA LAC program, and the lower part defines the values to write into OCLA configuration registers before the LAC program would be run. The .c file is the test, an Anthrax program to be loaded into PS-0 and PS-3 (in this case), which will configure OCLA registers, load the LAC program, start the LAC program, and create appropriate Ice9 activities so that this particular OCLA configuration and LAC program will trigger-on and collect interesting data.

11.17.3 Use Our Examples on a Real Machine

The OCLA configuration of any **<project>/sw/anthrax/tests/ocla/** simulation test can easily be converted into a diagnostics dash perl script for use on a real machine.
Instructions for how to convert the OCLA configuration (LAC program plus register configurations) of any of these simulation tests into a diagnostics dash script are found in **<project>/sw/anthrax/tests/ocla/README**, and consists of a quick make command. What you do is go to **<project>/sw/anthrax/tests,** the directory above where the tests are, and type "make ocla/<base_name>_cfg.pl", where <base_name> is the part of the filename ending in _util.cpp that's before the _util.cpp. The resulting perl script shows up in the **<project>/sw/anthrax/tests/ocla/** directory.

If you need an OCLA configuration different than what's found there, find one of the *_util.cpp files that's close to what you need, copy it to a new <something>_util.cpp name, change it to do what you want, and run the make command to get a dash script.

11.17.4 Create Your Own Counter

You can use OCLA as 1 or 2 highly-configurable counters.

In this use of OCLA, Collector Blocks are unused, CollectTrace never turned-on. A LAC program is needed, but it's fairly simple for the one-counter case. OCLA's two counters are in the LAC unit, incremented by LAC program instructions.

To create one counter, configure a trigger from any signal or combination of signals, and then write a LAC program that has a tight 1-state loop that increments the counter whenever the trigger is asserted. This counter has 12-bits in Ice9A, 16-bits in Ice9B.

To create two counters in Ice9B, configure 2 triggers, and have that 1-state tight loop increment one counter if one trigger is true, the other counter if the other trigger is true, and both counters if both triggers are true.

Creating two counters in Ice9A is less accurate, because Ice9A doesn't have the INCRBTH instruction, so if both triggers are true, you can only increment one of the counters.

To get one larger counter you can effectively concatenate the two available counters by having nested loops in the LAC program. This gives you 24 bits in Ice9A, 32 bits in Ice9B. When you nest the 2 counters there's a chance of tiny inaccuracies in the count because the LAC program has to ignore a potential event when clearing the lower counter, each time the lower counter rolls over and increments the higher counter.

11.17.4.1 You might prefer SCB Performance Counters

Because counting in OCLA requires a LAC program, it may be easier to feed the signals or triggers to SCB Performance Counters, and do the counting there. SCB Performance Counters are 32 bits whereas OCLA counters are only 16 bits (12 bits in Ice9A).

SCB Performance Counters is pretty powerful. If you wish to count one trigger qualified by another, SCB Performance Counters can do that. If you wish to count one trigger qualified by a delayed or advanced version of another trigger, SCB Performance Counters can do that, with the delays being applied in OCLA LAC before the triggers are sent to SCB Performance Counters.

One motivation to count in OCLA rather than SCB Performance Counters is that SCB Performance Counters has black-out periods (missing counts) whenever an SCB write or read is in progress.

Another motivation to create a counter in OCLA is if SCB Performance Counters is already in use, or if you wanted more than 2 continuously-counting counters. 2 continuous full-count counters in SCB Performance Counters plus one in OCLA gives you 3 at once.

2 in SCB Performance Counters plus 2 in OCLA gives you 4 at once.

11.17.5 Defensive Programming

Sometimes when you use OCLA on an Ice9 you don't know how that OCLA was used previously. State can be left around that will confuse the results of your OCLA run, or even interferes with it's operation! Even the same OCLA config-and-run done twice in a row can have problems the 2nd time you do it. Don't rely on reset values of any OCLA register in LAC, Trigger Blocks, or Collector Blocks. Reset is often long ago, with much history since.

Do one or both of: (a) Before your OCLA run, run an OCLA-config and LAC-program specifically designed to clean up everything. (b) Code your OCLA config and LAC program "defensively", to clean up everything it can in the beginning, as it gets started.

Here's a list of things to clean up before or during your config and LAC program, with "when to clean it up" in parentheses.

- **LAC Flag-0** (early LAC)
- **LAC Flag-1** (early LAC)
- **External OCLA trigger output pin** (early LAC)
- **LAC Debug Interrupt** (config before)
- **LAC Slow Interrupt** (config before)

- **all LAC Mask and Match registers, used or not** (config before)
- **every-CTB's EnableCollect** (config before)
- **every-CTB's Write Address** (config before)
- **you CTB is stuck-at-full** (config before)
- **every-CTB's contents** (separate OCLA run before)
- **CollectTrace** (separate OCLA run before, if needed)

All of these could be accomplished by a separate "cleanup" OCLA config-and-run, but most of the cleanup can just be included as part of the OCLA config-and-run you are writing for your desired purpose.

"early LAC" means clearing these in the first few instructions of your OCLA LAC program.

"config before" means during the SCB-registers configuration you must do to get ready to run your LAC program.

"separate OCLA run before" means doing a generic "cleanup" OCLA run, involving SCB-registers configuration, loading a LAC program, running that LAC program, maybe followed by more register writes.

Well-written LAC programs, properly manually-terminated if they don't see their trigger, do not leave the **CollectTrace** signal ON afterwards. But if it's ON, you may need or want to shut it OFF.

11.17.6 CTB stuck-at-full

From trial and error we've found it best to write the appropriate R_CtbxColCtb ***twice*** for each CTB you are using, otherwise you risk not collecting anything.

First write: EnableCollect=0, WtAddrClr=1, ExtMuxSel=your_desired_mux_setting, QTrigState and QualTrig = your_desired_settings, StopOnFull doesn't matter.

Second write: EnableCollect=1, WtAddrClr doesn't matter, ExtMuxSel=your_desired_mux_setting, QTrigState and QualTrig = your_desired_settings, StopOnFull=your_desired_setting.

11.17.7 Shutting-Off CollectTrace

In Ice9A chips, sometimes lac_ctb_CollectTrace_c2a gets left on. That can cause problems reading CTBs, and problems with the next OCLA run.

This is fixed in Ice9B and later to have a shut-off of the LAC program also shut-off CollectTrace.

CollectTrace can only be turned ON or OFF by a SETCOLL or CLRCOLL opcode in a running LAC program. In Ice9A there are no register writes which can turn it OFF, although a reset of the chip will turn it OFF.

All LAC programs should make sure to do a CLRCOLL before reaching their final state, or in their final state, no matter whether whether they have a "good" termination or "bad" termination (like a timeout, or user-requested termination).

11.17.7.1 Why would CollectTrace be Left ON?

CollectTrace can be left ON due to a bad LAC program, a LAC program with no timeout that never got a trigger, or by writing GO=0 to stop a LAC program in the middle, when CollectTrace is still ON. In Ice9A, now only a running LAC program that executes opcode CLRCOLL can shut it off!

11.17.7.2 Why is CollectTrace ON a Problem?

If CollectTrace is still ON after running a LAC program:

1. You may get all-zeros when reading-out the contents of a CTB! (even though the CTB does not contain all-zeros) Although misleading and frustrating, this can be solved by clearing that CTB's EnbleCollect bit.
2. As you start configuring for your next OCLA LAC program, some or all of the space in your CTB may get used-up before you can even say GO to your new LAC program! This applies when using a CTB in StopOnFull mode.

11.17.7.3 Is CollectTrace ON?

To find out if CollectTrace is ON, read bit "Collecting" in the R_CtbxColCtl of any CTB (even if that CTB has EnableCollect=0, or even even if it has StopOnFull=1 and full).

11.17.7.4 How to Read CTB Contents While CollectTrace is ON

Clear bit EnableCollect in the CTB's R_CtbxColCtl, then read the CTB.

11.17.7.5 Fastest Way To Shut Off CollectTrace in Ice9A

1. Write 0x00000000 to R_LacCtl.
2. Write 0x00000000 to all 5 Aggregate Mask Registers, R_LacAggMsk[4:0].
3. Write 0xffffffff to all 5 Aggregate Match Registers, R_LacAggMat[4:0].
4. Write 0x007 to R_LacRam[0x000], R_LacRam[0x400], R_LacRam[0x800], R_LacRam[0xc00]. (This is a tiny LAC program. There is no need to write or clear the other LAC locations.)
5. Write 0x00000001 to R_LacCtl.
6. Write 0x00000000 to R_LacCtl.

This should clear CollectTrace.

Of course you've now slightly messed-up your previous LAC program and previous OCLA registers configuration. You can either try to restore the changed values or load a complete new configuration and program for OCLA.

To restore: Prior values of R_LacCtl, R_LacAggMsk[4:0], and R_LacAggMat[4:0] could be read and remembered ahead of time, then restored afterwards. R_LacRam is write-only, so to know what values to restore to it you'll have to read your LAC-program source-code, or look at a logfile.

How this shuts-off CollectTrace:

The value 0x007 in R_LacRam[0x000] means {CLRCOLL, GO TO State-0}. The instruction in R_LacRam[0x000] will get executed by the write of 0x00000001 to R_LacCtl, and then CollectTrace will be OFF.

The other writes are to keep CollectTrace OFF during the time it takes to write 0x00000000 to R_LacCtl. Many LAC steps may get executed during that time. State-0 has 128 locations in LacRam, depending on Aggregate Match and counter overflow bits. The writes to Aggregate Mask and Match registers will zero-out the Aggregate Match bits, Reducing State-0 to only 4 locations based on counter overflow bits. With an 0x007 in all 4 of those locations we'll stay within those 4 locations, and not start executing other instructions of the previous LAC program (which might contain a SETCOLL).

Chapter 12

Clocking, ECC, Test Logic, Reset, and Initialization

[$Id: chipmisc.lyx 50689 2008-02-07 15:05:46Z wsnyder $]

12.1 Overview

This section describes the "miscellaneous" pieces of the ICE9 chip. These include:

- Clock generation and distribution
- ECC general description
- The Design For Test (DFT) support for internal test scan and boundary scan at manufacturing.[1]
- Reset and related logic
- Boot time-line

12.2 Differences, Bugs, and Enhancements

12.2.1 Product and Chip Pass Differences

1. ICE9A1 returns a different revision (ICE9A1 vs ICE9A0) when reading the IDECODE register.
2. ICE9B fixes Sms Reset syncronized to the wrong clock, bug2055. This required the smsclock to be turned off whenever we wiggle reset, then turned on again a bit later.
3. ICE9B eliminates R_SysTapDint, replaced with the SCB-space R_ScbDInt, bug2223.
4. ICE9B supports transmit interrupts for R_SysTapAtnMsp, and separates RW1C bits, bug2222.
5. NEED IMPL: TWC9A changes the default value for R_SysTapPll_D*clkDifv to support a processor default clock frequency of *FIX* MHz, bug3384.
6. TWC9A fixes access to any SCB bus slave hanging while the DDR controller is in reset, bug2928.
7. TWC9A adds an R_SysTapReset_Lac and _Pmi to separate the R_SysTapReset_Scb bit from also controling the BBS/PMI reset, bug2929. Earlier products needed caution when maintaining FSW/FL traffic during partial reboots.
8. NEED IMPL: TWC9A adds R_SysTapReset_Proc6, and _ProcSms6 to support the additional cores.
9. TWC9A uses R_SysTapInstrTwc instead of R_SysTapInstReg to support the additional cores.

[1] See also the "IEEE Standard Test Access Port and Boundary-Scan Architecture" ref. document; IEEE Std 1149.1-2001 IEEE Joint Test Action Group (JTAG).

10. TWC9A adds R_SysTapScb64 to access doubleword SCB registers. Code should use this new registers or 64 bit SCB registers will not be visible.
11. NEED IMPL: TWC9A adds R_SysMemInit register and associated functions for on-chip memory initalization. In previous products BIST was used to initalize on-chip memories.
12. NEED IMPL+SPEC: TWC9A will merge the SysChain and E-Silicon chain on-chip instead of off-chip.
13. NEED IMPL+SPEC: TWC9A will replace or make the E-Silicon chain IEEE compliant (on the correct edges).

12.2.2 Known Bugs and Possible Enhancements

1. [Larry] Add a new LBS+SCB region. The msp could set the start address in 32 or 64 bit steps, and then scan in, say 128 bytes with a continuous shift on the scan. Then, while the ice9 digests that block, the msp scans in 128 bytes into the alternate half of the block. This is essentially a block of shared memory accessed on the ice9 side by scb and on the msp side by efficient scan. The scan chain would shift in a direction compatible with the qspi as well. This shared area would be used instead of fastdata (since it would be much faster) for boot2 loading, and we would also use it for block transfers of attn data instead of doing that 26 bits at a time via the current attn register.

12.3 Clock generation and distribution

12.3.1 Goals and Features

The Sicortex system clock architecture (includes specifics of board design) has the following goals:

1. The system clock architecture has one system clock (sys_clk) and each board receives a copy of the sys_clk. The system clock architecture will minimize the possibility of a single point of failure in the clock tree.
2. The distribution frequency of the system clock (sys_clk) will be 66.67 MHz and with a long term accuracy of 100 ppm, and jitter spec of +/- 50ps.
3. Each ICE9 chip will generate on-chip clocks for its sub-systems using 2 (differential) copies of sys_clk (sys_clk_e_h/l & sys_clk_o_h/l). Thus all generated clocks in the system will be derived from a single oscillator.
4. The inter-ICE9 fabric is a “Mesochronous Interconnect” where each node in the fabric is frequency locked (but not phase locked) with every other node.
5. The fabric switch operating speed is targeted at 200MHz. Correspondingly, the fabric link will operate at 8B10B encoded data rate of ten times the operating speed of the fabric switch. The PLL design will allow adjusting fabric switch clock speed by up to +/- 25% from its design goal.
6. The primary design goal of the processor/cache operating speed is 500MHz/250MHz. The PLL design will allow selecting processor/cache clock speed by as much as +/- 20% from its design goal.
7. The primary design goal of DDR2 interface is to operate with industry standard SDRAM DIMMs. The industry standard SDRAM are (will be) available at 200/266/333/400 MHz clock speeds. The PLL design will allow DDR2 clock speed selection from 200 MHz to 400 MHz.
8. The primary design goal of PCIe root complex and PCIe controller is to use clock at 250 MHz. The primary design goal of PCIe PHY is to use RefClk clock at 125 MHz. These clocks come from the PCI Express PHY. The PLL design will generate PCI reference clock at 100 MHz for use by the PHY and to be driven off-chip for use by an attached card.
9. The PLL design will allow configuring each PLL in BYPASS mode. (See the test clock discussion in Section ????.)

Clock generator features of ICE9 are listed below:

1. ICE9 clock domains can be categorized into four clock groups as follows:

(a) Group-A: For Fabric switch and fabric links, sclk from 200MHz to 250MHz.

(b) Group-B: Processor/cache clocks, pclk/cclk maintaining phase aligned 2:1 frequency ratio for pclk and cclk. The range of pclk is from 400 MHz to 800 MHz.

(c) Group-C: DDR2 clocks, dclk. This group will need dclk and dclk90. The operating range of dclk is from 250 MHz to 400 MHz. Operating values are 200, 267, 333, and 400MHz. Each of the the two DDR2 interfaces has it's own PLL to generate the in-phase and quadrature clocks: d1clk & d1clk_m90 and d0clk & d0clk_m90.

(d) Group-D: PCIe interface, pci_ref_clk/pci_ref_clk_x2 maintaining a 1:2 frequency ratio, phase alignment is not necessary, for pci_ref_clk and pci_ref_clk_x2 at 100 MHz and 200 MHz.

2. ICE9 will use one PLL design, (called PLL_AB), to generate clocks for various sub-systems.
The PLL_AB design has two outputs. The relationship bewteen the two outpus is configurable from three choices. The output selection choices are :

   (a) DIV2-0deg, DIV4-0deg : factor of 2 frequency difference, outputs are phase aligned.

   (b) DIV4-0deg, DIV4-90deg : same frequency, 90 degree phase shift between outputs

   (c) DIV4-0deg, DIV8-0deg : factor of 2 frequency difference, outputs are phase aligned.

3. ICE9 has total of five instances of the PLL_AB design. The 5 PLLs are placed in 2 groups: one near the south-west (odd-link) corner of the chip and one near the north-east (even-pci) corner of the chip. The east-side PLL group contains the pclk/cclk PLL, the pci_ref_clk PLL and the d0clk/d0clk_m90 PLL. The west-side PLL group contains the sclk PLL and the d1clk/d1clk_m90 PLL.

4. ICE9 will get 2 copies of the differential sys_clk on 4 reference-clock input pins. The "RefClk" pin of all 5 instances of the PLL_AB will be connected to the sys_clk nearest it.

12.3.2 Sys_clk distribution tree

The Sicortex system will use a backplane and connectors as the inter-board connection medium. The backplane will not have any active components. Boards make signal connections to each other through its connector on the backplane.

In the chassis, the clock distribution tree originates at an oscillator operating at 133.33 MHz. The oscillator output is divided by 2 and then distributed as "sys_clk" at 66.67 MHz to all boards. On board, a copy of sys_clk is connected to the 2 "sys_clk" inputs of each of the 27 ICE9 chips. Because all copies of sys_clk in the chassis originate from a single oscillator, all generated clocks in ICE9 are frequency locked w.r.t. to each other. The sys_clk input to ICE9 is in 2 distinct pairs of LVDS pins received in 2 LVDS receivers - one for the southwest PLL group and one for the northeast PLL group. The board-level sys_clk distribution tree has 54 sys_clk destinations on each module board (2 for each ICE9 chip).

The system clock distribution scheme is shown in Figure-12.1.

Figure 12.1 shows three connectors, M, N, and P, each receiving copy of sysclk and driving buffered version of the sys_clk to 27 ICE9 chips with 2 receiver ports each. The on-chip clock generating in ICE9 consists of 5 instances instances of PLL_AB.

Figure 12.2 shows that in ICE9, the clock PLLs generate clocks for Processor/L2-cache, DDR2 interface, PCIe interface, and the fabric switch. The fabric switch clock, in conjunction with multiple fabric link receiver PLLs, and multiple fabric link transmitter PLLs, builds the complete clocking scheme of the fabric links. The fabric link clocking is described below. A similar strategy is employed for the PCIe SERDES links.

Each fabric link connects two logically adjacent ICE9 chip using SERDES PHY technology which drives embedded clock and data on differential pair of wires. The sclk PLL generates the clock for fabric switch which is which is also used by the link transmitter PHY. The fabric link transmitter PHY has a PLL, called Tx_PLL (an integral part of the link PHY), which uses the fabric switch clock signal as a reference clock and drives a serial data stream on the transmitter PHY port five times faster than the switch clock in DDR mode. The fabric link receiver also has a clock-data-recovery PLL (CDR-PLL), also integral to the link PHY and dedicated to the receive lane, to recover data and clock from incoming data streams.

In Figure 12.1, five instances of the PLL_AB will use sysclk at 66.67 MHz as a reference clock which is sourced from single oscillator, hence, all generated clocks will operate in frequency locked mode w.r.t. each other.

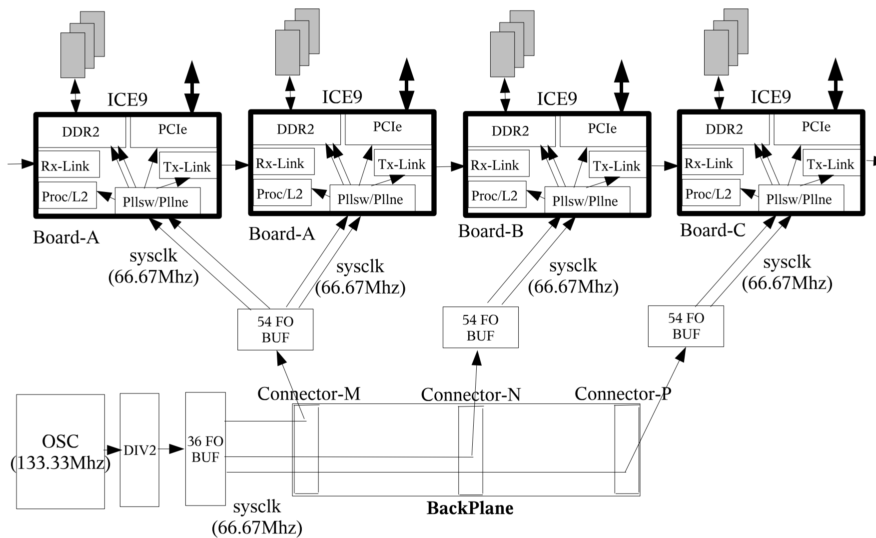


Figure 12.1: Clock Tree Distribution

Note-: Jitter spec (estimated, needs validation) of sysclk at the pins of ICE9

Instance	ppm spec	jitter spec in ps
OSC : 133.33 MHz	100	25
Divider : DIV2	-	TBD
BUF : 36_FO	-	TBD
BUF : 27_FO	-	TBD
sysclk @ ICE9	100	+/- 50

12.3.3 Clock Generation in ICE9

The clock generation for ICE9 takes place in 2 physically distinct PLL groups. For logical purposes these may be treated as a single module, though the chip hierarchy will include them as separate entities. The logical clock generation module is shown in Figure 12.2. It has five instances of the PLL_AB and it generates sclk for the fabric switch interface, pclk/cclk for the processor core and L2-cache interface, separate dclks for the DDR2 controllers, and pci_ref_clk for the PCIe interfaces. Each instance of the PLL_AB has several control signals, described below. There are two instances of the PLL_AB for generation of the two dclks. Each dclk domain (d0clk & d1clk) will be provided with a "normal" clock signal (used for the majority of the logic) and a -90-degree phase clock (used only in the PHY).

12.3.4 PCIe clocking

The clocking scheme for the PCI express interface has changed from the original plan. The PLL originally planned to generate the 250MHz iclk will now generate a 100MHz pci_ref_clk from the 66.67MHz sys_clk. The 100MHz pci_ref_clk will then be driven off-chip to the clock pin of the PCIe slot on the module board (perhaps through a buffer or level translator). It will also be driven to the PCIe PHY, where it will be used to generate the 250MHz iclk (and internally to clock the SERDES transmitters). The result is that the root of the iclk tree will now be an the output pin of the PCIe PHY.

Note that the PCI Express specification allows the reference clock frequency to be "downspread" by up to 0.5%, to allow spread-spectrum clocking for radio-frequency emissions control purposes. The system design may take advantage of this by using more widely available 133MHz oscillators, resulting in a 66.5MHz sys_clk frequency, 0.25% below nominal. This works because both ends of all our PCI Express links will use the same reference clock

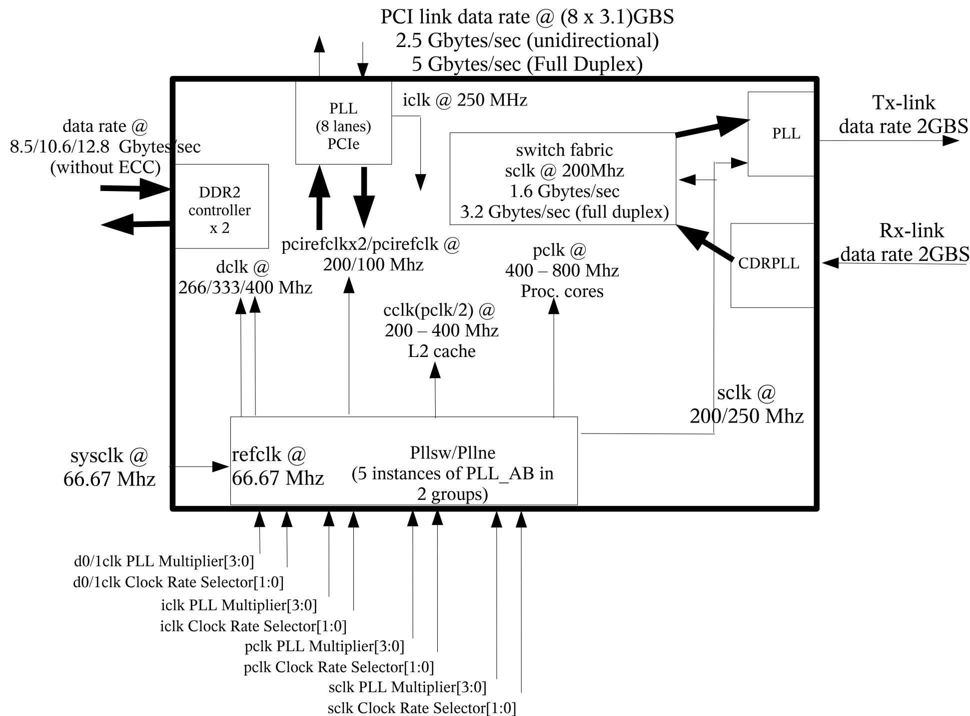


Figure 12.2: ICE9 Clocks and Data Rates

as just described.

12.3.5 Block diagram of PLL_AB

The block diagram of the PLL_AB is shown in Figure 12.3 and the pins are listed in Table 12.1.

The PLL receives REF input as its reference clock input and its VCO multiplier factor through DIV signal. The PLL_LOCK signal is a status signal which will be set when PLL has acquired lock. The PLL can be held in reset state by RESET signal.

There are 2 outputs from PLL_AB. They are PLLOUT-1 and PLLOUT_2. Both outputs from PLL are configurable through OUTPUT_SEL signal. There are 3 choices of output selection.

The PLL_AB also supports PLL in bypass mode when BYPASS_ENAB signal is set. In bypass mode, there are 2 options available for selecting BYPASS_CLK at two output ports - (a) Both outputs are connected to BYPASS_CLK, and (b) One of the outputs is connected to the half frequency clock of BYPASS_CLK.

ICE9 PLL Instantiation & Configuration Notes:

1. The RESET signal for the PLL_AB must be gated with a decode of {test_mode_en, test_mode[*]} to ensure it is asserted in the appropriate scan modes.

2. All pins (including REF signal) of PLL_AB are regular core-voltage CMOS signals.

3. Control signals for the PLL_ABs which are CSRs must be explicitly registered on the appropriate chain. The PLL macro does not register the bits internally.

4. Any changes to CSR bits affecting PLL operation should be appropriately guarded by reset for both the PLL and downstream (clocked) logic to prevent deleterious effects due to unstable PLL operation, clock glitches, runt pulses, etc.

5. Invalid settings: When DIVF[4:0] is less than 5'd11 or greater than 5'd23 or OUTPUT_SEL[1:0] equals to 2'd3, no damage will occur to the PLL, but the output behavior is not defined.

Signal Name	From	To	Description
REF	primary input	PLL_AB	Reference clock at 66.67 MHz
RESET	SysChain	PLL_AB	PLL internal reset. This signal is gated with scan_enable and stays asserted during chip reset.
DIVF[4:0]	SysChain	PLL_AB	VCO feedback divider encodings of 4'd11 through 4'd23 will provide multiplier from 12 to 24. Multiplier value = (DIVF[4:0] + 1)
OUTPUT_SEL[1:0]	SysChain	PLL_AB	PLL output selector for [PLLOUT_1, PLLOUT_2]. The selector encodings are: 0 - DIV2, DIV4 (both outputs are phase aligned) 1 - DIV4, DIV4-90 2 - DIV4, DIV8
BYPASS_ENA	SysChain	PLL_AB	PLL bypass enable
BYPASS_CLK1	primary input	PLL_AB	Bypass clock when BYPASS_ENA is set
BYPASS_CLK0	primary input	PLL_AB	Bypass clock when BYPASS_ENA is set
BYPASS_CLK_SEL	SysChain	PLL_AB	selects BYPASS_CLK0 or 1 when for output BYPASS_ENA asserts
LOCK	PLL_AB	SysChain	PLL Lock indicator
PLLOUT_1	PLL_AB	clock-tree	PLL_1 output. This signal has 50% duty cycle in normal mode. Refer to encodings of OUTPUT_SEL[1:0]
PLLOUT_2	PLL_AB	clock-tree	PLL_2 output This signal has 50% duty cycle in normal mode. Refer to description of OUTPUT_SEL[1:0]
Analog VDDA/VSS	chip bumps	PLL_AB	Analog power and ground pins (chip bumps)
VDD/VSS	I/O ring	PLL_AB	Core power/ground, connect by abutment in the I/O ring

Table 12.1: PLL_AB Pins

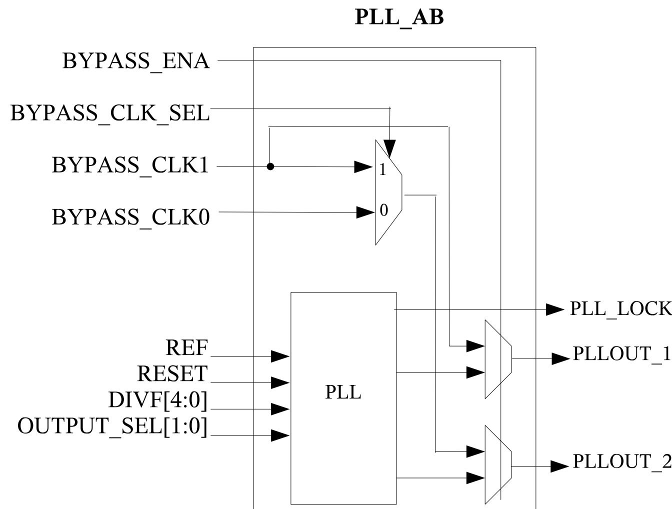


Figure 12.3: PLL_AB Block Diagram

BYPASS_ENAB	RESET	BYPASS_CLK_SEL	PLL_LOCK	PLLOUT_1	PLL_OUT_2
0	0	X	Normal mode	Normal mode	Normal mode
0	1	X	0	LOW	LOW
1	X	0	0	BYPASS_CLK1	BYPASS_CLK0
1	X	1	0	BYPASS_CLK1	BYPASS_CLK1

Table 12.2: PLL Bypass Control

6. The divider flops, including “DIV2” flop on BYPASS_CLK path, are not scannable. If they do not work it will become apparent when no clock is observed.

12.3.5.1 Bypass mode in PLL_AB

Each PLL_AB has three primary pins to support bypassing PLL. Those pins are BYPASS_ENAB, BYPASS_DIV2_ENAB, and BYPASS_CLK. The output of the PLL_AB will be selected as per Table 12.2. The pins are driven by the test mode controller based on the state of the test mode pins described in Table 12.4 and by the SysChain scan control chain that is used by the module service processor to initialize and configure the ICE9 chip. (See Section 12.6.9.)

12.3.6 Implementation of PLL_AB

ICE9 will have five instances of PLL_AB to generate primary clocks - sclk, pclk, dclk, and pci_ref_clk. (There are 2 instances of the DDR clock PLLs for the 2 dclk domains.) The clock implementation is shown in Figure-12.4.

The implementation scheme provides range of operating speeds for each clock by varying DIVF[4:0] input.

Valid settings and the range of clock outputs for those settings are shown in Table 12.3.

Note that the first row identifies clock name and the value of OUTPUT_SEL[1:0] pins in brackets. This register is controlled via the SysChain scan registers described in Section 12.6.9.

The 5 PLLs are placed on the chip in 2 groups: Pllsw & Pllne. Pllsw contains an LVDS sys_clk receiver, PLLs for d1clk/d1clk90 & sclk, and an LVDS driver for test_clk_o_h/l. Pllne contains contains an LVDS sys_clk receiver,

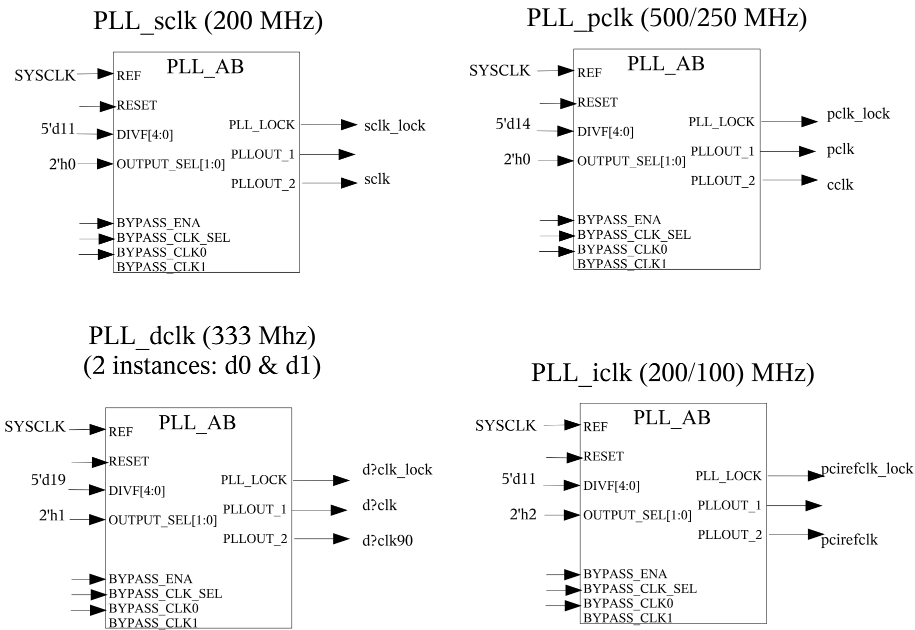


Figure 12.4: Clocks using PLL_AB

DIVF[4:0]	sclk (osel=0)	sclk (osel=2)	pclk/cclk (osel=0)	dclk (osel=1)	pci_ref_clk_x2/pci_ref_clk (osel=2)
5'd0 - 5'd10	invalid	invalid	invalid	invalid	invalid
5'd11	200	100	400/200	200	200/100
5'd12	217	108	433/217	217	217/108
5'd13	233	117	467/233	233	233/117
5'd14	250	125	500/250	250	250/125
5'd15	267	133	533/267	267	267/133
5'd16	283	142	567/283	283	283/142
5'd17	300	150	600/300	300	300/150
5'd18	317	158	633/317	317	317/158
5'd19	333	167	667/333	333	333/167
5'd20	350	175	700/350	350	350/175
5'd21	367	183	733/367	367	367/183
5'd22	383	192	767/383	383	383/192
5'd23	400	200	800/400	400	400/200
5'd24 - 5'd31	invalid	invalid	invalid	invalid	invalid

Table 12.3: PLL VCO Scaling Factors

PLLs for d0clk/d0clk90, pci_ref_clk, & pclk/cclk, and an LVDS driver for pci_ref_clk_l/h.

The test_clko_o_h/l and the pci_ref_clk_h/l LVDS output pins are driven through muxes to select several operational and test clocks as indicated in section 12.6.9

12.4 General ECC strategy

This section on ECC strategy describes general guidelines for implementation of ECC on the ice9 chip. Specifics of how the ECC is implemented in any given section are described in the appropriate chapter of this spec.

The following registers should be implemented by memories which have ECC generation and/or checking. All of these registers are read/write master/slave registers on the SCB (or other software-visible bus/chain). Access to SCB registers and operation of the SCB is described in the "Serial Configuration Bus" chapter of the chip spec. The specific names of these registers is documented with each section's SCB registers.

Control Registers		**Status Registers**
ECC_Mode_Register[1:0]		ECC_Error_Status_Register[2:0]
ECC_Drive_Bad_Data_Register[1:0]		ECC_Error_Address_Register[x:0] (not all cases, see below)
		ECC_Error_Syndrom_Register[7:0] (not all cases, see below)

ECC handling for the L1 caches (I & D) has been modified to leverage the existing parity and interrupt mechanisms in the M5Kf processor core and is therefore somewhat differenc than described here. The L1 I-cache treats a parity error as a miss, which causes a fetch from the (ECC protected) L2 cache. This effectively provides single-bit-error correction but not double-bit-error detection. The L1 D-cache implements byte-wide ECC to support byte writes. See the Processor Segments chapter for more details.

12.4.1 ECC Control Register descriptions:

12.4.1.1 ECC_Mode_Register[1:0] (associated with ECC correction)

ECC_Mode_Register[1] - ECC error detection enable: Enables Writing of ECC status registers and assertion of the ECC interrupt line from this block.

ECC_Mode_Register[0] - ECC error correction enable: Enables ECC correction of data passing through the correction block

12.4.1.2 ECC_Drive_Bad_Data_Register[1:0] (associated with ECC generation)

ECC_Drive_Bad_Data_Register[1] - flip bit [1] of the *data* coming out of the ECC generator (into the storage array)

ECC_Drive_Bad_Data_Register[0] - flip bit [0] of the *data* coming out of the ECC generator (into the storage array)

Asserting either causes a single-bit error to be generated. Asserting both causes a double-bit error to be generated.

Note:

- In most cases, "ECC_Drive_Bad_Data_Register" applies to all writes after the bit(s) are set, relying on software restrictions (i.e., clearing the register bit at an appropriate time) to ensure that reasonable behavior is obtained during software testing.
- If convenient, "ECC_Drive_Bad_Data_Register" MAY be implemented as a single-cycle operation (i.e., only the first write after asserting bits in the register contains bad data; then the register bit is cleared & subsequent writes return to normal operation).

12.4.2 ECC Status Register Descriptions

12.4.2.1 ECC_Error_Status_Register[2:0] (associated with ECC correction)

ECC_Error_Status_Register[2] - sets if more than one ECC error occurs, i.e, if (ECC_Event_Occurs && ECC_Mode_Register[1] && (ECC_Error_Status_Register[1] || ECC_Error_Status_Register[0]) => set ECC_Error_Status_Register[2]

ECC_Error_Status_Register[1] - sets if an ECC-correctable error is detected

ECC_Error_Status_Register[0] - sets if a non-correctable ECC error is detected

Note:

- Updates of ECC_Error_Status_Register due ECC errors are blocked if ECC_Mode_Register[1] is deasserted.

For ECC correctors on the path to/from main memory (i.e., coming on/off the CSW), the following 2 registers may also be required:

12.4.2.2 ECC_Error_Address_Register[x:0] - x depends on the size of address space (associated with ECC correction)

Holds the (physical) address of the first ECC error since setting of any bit of ECC_Mode_Register[1:0]. This register is required only for ECC checkers for data on the main memory path in ICE9 (i.e., at the CSW interfaces to the L2 caches in the processor slices, and, optionally, at the Pci/Csw interface and at the Dma/Csw interface.)

12.4.2.3 ECC_Error_Syndrom_Register[7:0] (associated with ECC correction)

Holds the syndrome of the first ECC error since setting of any bit of ECC_Mode_Register[1:0]. This register is required only for ECC checkers for data on the main memory path in ICE9 (i.e., at the Csw interfaces to the L2 caches in the processor slices, and, optionally, at the Pci/Csw interface and at the Dma/Csw interface.)

Note:

- bits of ECC_Error_Status_Register & ECC_Error_Address_Register are set by the ECC logic during operation. Clearing of the register bits following an ECC event is up to software as a part of the interrupt routine triggered on a ECC event.
- Separate ECC_Error_Status_Register, *_Address_Resister and *_Syndrom_Register will be required for data coming out of the L2 cache and for data coming out of the CSW to distinguish between ECC events in the L2 and events in the CSW/DDR memories.

12.4.3 ECC Implementation & Test considerations

In order to test the ECC logic during manufacturing chip test, we'll need to ensure observability of the outputs of the ECC generation logic and controllability over the inputs to the ECC correction logic. If we don't do anything special this is a problem because the whole point of ECC is to transparently correct errors without impacting normal operation. So, what we're doing is the following:

12.4.3.1 Compiled memories with Synchronous Write Through (SWT) mode

When the Virage compiled memory supports SWT, we'll use it. With appropriate control settings, SWT provides a path for the write data coming into the memory to bypass the array and instead go to a flop, which is then driven (through a mux) to the output pins. The additional logic is incorporated in a wrapper around the memory array. The added flop is on a scan chain with control signals, scan-in and scan-out brought to pins of the wrapper. See Figure 12.5

The BypassMUX and OutputMUX select signals must be set appropriately during test (by tying them to a decode of test_mode). Once that's done, ECC generator outputs become observable via the scan flop and it's scan chain. Controllability over the inputs to the ECC correction logic is accomplished via the same mechanism. Nothing special is required in the design of the logic around the RAM.

12.4.3.2 Compiled memories with Asynchronous Write Through (AWT) and no Synchronous Write Through (SWT)

For this case, there are 2 concerns: 1) observability and controllability for testing the ECC logic, and 2) ensuring that AWT does not introduce combinational loops. Since the compiled memory does not provide a convenient scan-flop and we'll need to provide one externally (”rammaker” will be modified to do this by default). We have a choice of putting the mux on the memory inputs or outputs; to be consistent with what's provided for SWT-enabled rams, we'll put in on the output, unless there's a reason not to. If necessary, the flop and mux may be inserted into upstream of the RAM on the data input side of the compiled memory & wrapper; see Vasu about a change to rammaker if you need to do this. (See Figure 12.6.) The OutputMUX select signal should be tied to a decode

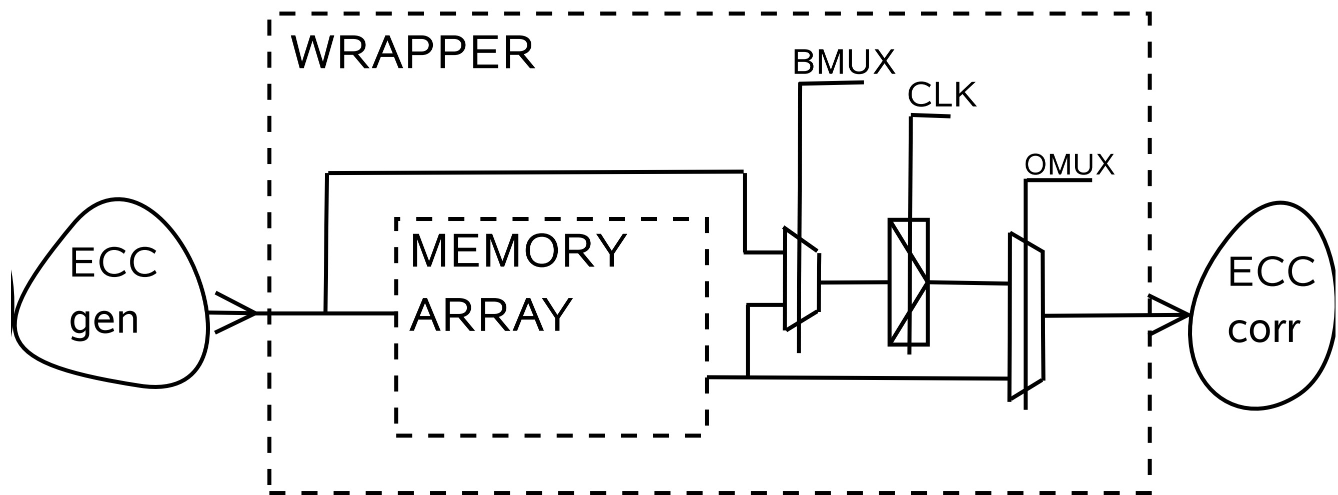


Figure 12.5: SWT ECC observability & controllability

of test_mode, as should the BypassMUX select signal. In this case the OuputMUX and the flop must be explicitly incorporated into the design. Scan insertion of the flop will ensure the the necessary observability/controllability is achieved. (If the instance of the RAM requires immediate flopping of read data before ECC correction, there is no need to add anything special; observability & controllability are already available.)

By default, the explicit flop & mux should be added to both data and ECC correction bits. If no combinational loops are introduced by AWT, the flop/mux may be added to only the ECC bits, thus saving on the flop count.

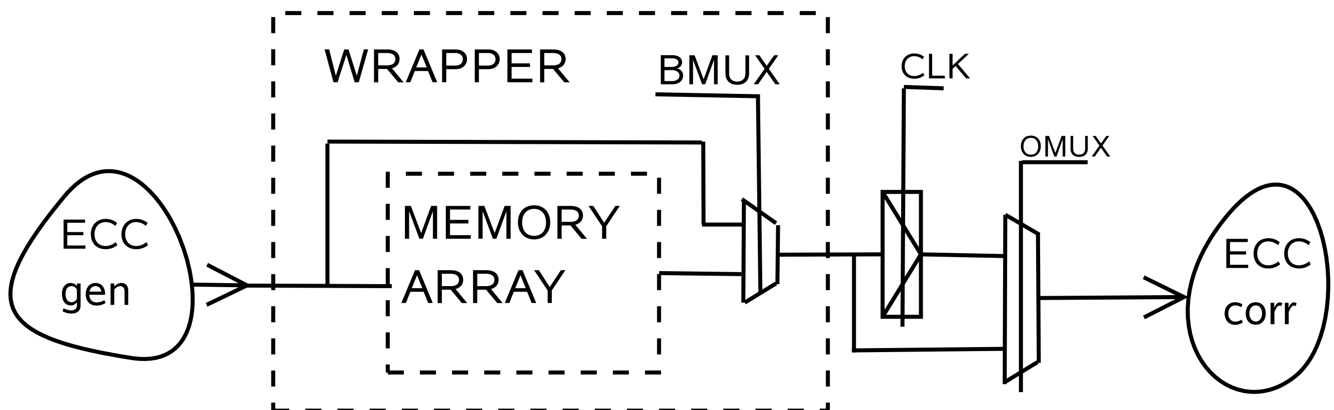


Figure 12.6: AWT ECC observability & controllability (also breaks any combinational loops)

12.5 DFT and Test Support

The ICE9 chip supports two different scan interfaces for test.

The first is a serial "muxscan" interface used for chip test at wafer and die test stages. It provides up to 100 parallel scan chains and test mode configuration pins. The scan modes are selected via the test mode input pins as shown in Table 12.4. The control pins relating to muxscan features are all prefixed with the name "test_"; any pin with the prefix "test_" is used in test-modes only and can be tied off for normal operation. As per eSilicon's practice, the test control pins are: **test_scan_en** (eSilicon's name is scan_enable), **test_mode_en** (eSilicon: chip_test), and **test_mode[2:0]** (eSilicon: test_mode!). When the ICE9 chip is installed on a module, test_scan_en and test_mode_en will be tied FALSE and the other three test_mode[2:0] pins will be ignored. In muxscan mode ("stuck-at scan" and "transition fault scan"), the DDR DQ & AD pins provide 88 bits of scan data output and scan data input between the two DDR controllers. The DDR DQ & AD spins also have test_sdi[*] & test_sdo[*] overrides. See Section 17.3 for a complete list of signal pins and test-mode overrides. The remaining 12 bits of scan in and scan out are provided on dedicated pins labeled test_sdi[99:88] and test_sdo[99:88]. Some of the entries

in Table 12.4 seem to be duplicates with respect to PLL bypassing. In some cases, they are assigned different test_mode[3:0] entries due to different test-mode overrides.

Anytime the PLL output is bypassed with a test_clk or syc_clk, that PLL should be held in reset by the LBS.

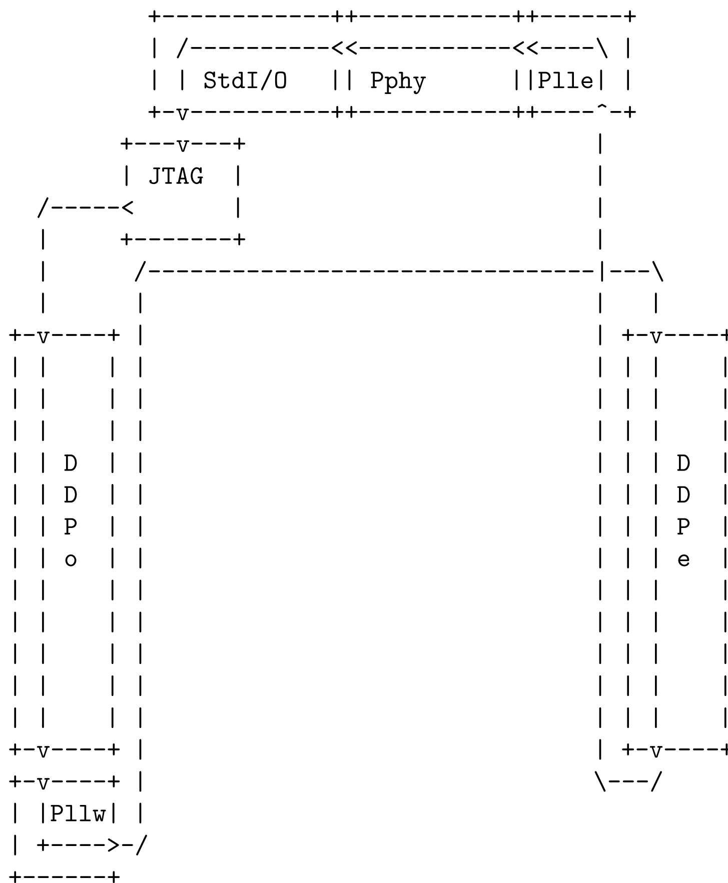
The second test interface is the JTAG test scan chain used for boundary scan. This mode is implemented in an IEEE-JTAG 1149.1 Test Access Port (TAP) controller supplied by eSilicon. The JTAG chain has its own chip pins, prefixed with "jtag_" and only these signals carry the jtag_ prefix.

The SysChain, described below, is used for in-system maintenance and initialization. The SysChain may also be used to set PLL controls and Virage RAM configuration parameters during manufacturing test.

Because specifics of the distribution of the clocks and reset signals is important to ATPG test generation, it's further described in Figure 12.10

12.5.1 Boundary scan (normal mode)

For board-level continuity testing, the chip supports JTAG boundary scan. The PCIe PHY comes as a hard macro with boundary scan pre-inserted. The link PHYs do not support boundary scan. The DDR I/Os, LVDS clock I/Os and selected general-purpose I/Os will have boundary scan cells inserted by eSilicon along with the JTAG TAP controller insertion. The boundary scan-chain ordering follows the diagram below (JTAG TAP -> DDPo -> Pllsw -> DDPe -> Pllne -> PCIe PHY -> general purpose I/O block -> JTAG TAP):



12.5.2 Stuck-at Scan (test mode 16)

eSilicon ATPG tests using mux-scan. Virage memories in SWT-mode (where supported) or AWT-mode.

12.5.3 Transition Fault Scan (test_mode 17)

Similar to stuck-at scan - eSilicon ATPG tests using mux-scan. Virage memories in SWT-mode (where supported). AWT-mode should not be used due the multi-cycle paths created.

test_mode_en, test_mode[3:0]	Description	pclk/cclk PLL	sclk PLL	pci_ref_clk (AB) PLL / iclk (PHY) PLL	d0clk/ d0clk_m90 PLL	d1clk/ d1clk_m90 PLL
0, X (0-15) (test_mode[*] will be tied low on the module board)	Normal Operation + (DDR, link PHY &, PCIe PHY functional tests w/ all PLLs) + (Memory BIST w/ PLL) + (JTAG BScan)	**operating**	**operating**	**operating/ operating**	**operating**	**operating**
1, 0 (16)	Stuck-at scan - Virage SWTon/AWTon	**test_pclk/ test_cclk**	**test_sclk**	**test_iclk/ bypassed**	**test_d0clk/ test_d0clk**	**test_d1clk/ test_d1clk**
1, 1 (17)	Transition Fault Scan - Virage SWTon/AWToff	**test_pclk/ test_cclk**	**test_sclk**	**test_iclk/ bypassed**	**test_d0clk/ test_d0clk**	**test_d1clk/ test_d1clk**
1, 2 (18)	PLL, separate pins?, low speed, lock	**operating**	**operating**	**operating/ operating**	**operating**	**operating**
1, 3 (19)	DDR ODT & drive strength parametric BScan	**operating**	**operating**	**operating/ operating**	**operating**	**operating**
1, 4 (20)	Memory BIST (no PLL)	**test_pclk/ test_cclk**	**test_sclk**	**test_iclk/ bypassed**	**test_d0clk/ test_d0clk**	**test_d1clk/ test_d1clk**
1, 5 (21)	DDR Functional Tests (no PLL)	**test_pclk/ test_cclk**	**test_sclk**	test_iclk/ bypassed (inactive)	**test_d0clk/ sys_clk_e (90deg apart)**	**test_d1clk/ sys_clk_o (90deg apart)**
1, 6 (22)	Slow DDR DLL Test (no PLL)	test_pclk/ test_cclk (inactive)	test_sclk (inactive)	test_iclk/ bypassed (inactive)	**test_d0clk/ sys_clk_e**	**test_d1clk/ sys_clk_o**
1, 7 (23)	Fast DDR DLL Test (PLL)	test_pclk/ test_cclk (inactive)	test_sclk (inactive)	test_iclk/ bypassed (inactive)	**operating**	**operating**
1, 8 (24)	PCI Functional Tests (PLLs)	**test_pclk/ test_cclk (slow, maybe inactive?)**	**test_sclk (slow, maybe inactive?)**	**operating/ operating**	**test_d0clk/ test_d0clk (slow, maybe inactive?)**	**test_d1clk/ test_d1clk (slow, maybe inactive?)**
1, 9 (25)	PCI Functional Tests w/o pci_ref_clk PLL (*PCIe PHY iclk PLL operating)*	**test_pclk/ test_cclk (slow, maybe inactive?)**	**test_sclk (slow, maybe inactive?)**	**test_iclk/ operating**	**test_d0clk/ test_d0clk (slow, maybe inactive?)**	**test_d1clk/ test_d1clk (slow, maybe inactive?)**
1, 10 (26)	PCI Functional Tests (no PLLs) (*PCIe PHY iclk PLL bypassed*)	**test_pclk/ test_cclk (slow, maybe inactive?)**	**test_sclk (slow, maybe inactive?)**	**test_iclk/ bypassed**	**test_d0clk/ test_d0clk (slow, maybe inactive?)**	**test_d1clk/ test_d1clk (slow, maybe inactive?)**
1, 11 (27)	Fabric link Transceiver Functional Tests (PLL)	**test_pclk/ test_cclk (slow)**	**operating**	**test_iclk/ bypassed (slow)**	**test_d0clk/ test_d0clk (slow)**	**test_d1clk/ test_d1clk (slow)**
1, 12 (28)	Fabric link Transceiver Functional Tests (no PLL)	**test_pclk/ test_cclk (slow)**	**test_sclk**	**test_iclk/ bypassed (slow)**	**test_d0clk/ test_d0clk (slow)**	**test_d1clk/ test_d1clk (slow)**
1, 13 (29)	UNUSED					
1, 14 (30)	UNUSED					
1, 15 (31)	UNUSED					

12.5.4 PLL Test (test mode 18)

A test of the 5 primary clock PLLs. With a 66.67MHz differential sys_clk_o & sys_clk_e,

- look for the lock indication from each PLL - present at the *clkLock pins (active in test-mode 18, see Section 17.3) or in the PLL control register (Section 12.6.9)

- Step through entries in Table 12.13, using the ClkOutCtrl[*] pins (active in test-mode 18, see Section 17.3) or the PLL control register, to bring out all possibilites listed. Depending on tester capabilities, check for presence of a toggling LVDS signal, check duty cycle, and check frequency.

12.5.5 DDR ODT & Drive Strength Parametric Test (test mode 19)

Similar to normal operation, boundary scan is used for parametric testing of the DDR PHY inputs & outputs with controllable drive strength (impedance) and controllable on die terminaion (ODT). In this mode the test_sdi[99:88] & test_sdo[99:88] pins are used to control the drive impedance settings, the ODT termination settings, and the ODT (read) termination enable for both instances of Ddp. In addition, the results of the impedance calibration block for the 2 instances of Ddp are available. See Section 17.3 for detail on test mode 19 pin overrides. *Because this test is performed with JTAG boundary scan functioning, the pins we overrride in this test mode must NOT be boundary scan inserted (or they may have observe-only boundary scan insertion).*

By making the results of the impedance calibration logic available at the chip pins, it is possible for the tester to check the impedance calibration using at least one and possibly several values of precision external resistor.

12.5.6 Memory BIST and Repair (test mode 0, 20)

Memory BIST is typ[ically done in the normal operating mode; bypassing PLLs with test_mode 20 is available if needed. This path uses the JTAG 1149.1 TAP controller to access the Virage STAR Memory self test and repair features. Two test-modes are provided, one with clocking from active PLLs, one with the active PLLs bypassed.

12.5.7 DDR Functional Test (test modes 0, 21)

It is expected that DDR functional tests will be done in normal operating mode. Test mode 21 is available if we want to bypass all PLLs for DDR functional testing. DDR functional tests probably require code running on a M5Kf core - specifics open here pending recommendation from the eSilicon DDR design team. We may need some pretty fancy load board design to support full-speed testing of the DDR I/Os.

12.5.8 Slow DDR DLL Test (test mode 22) (whether all DLL tests will be used in mfg. test is still open)

Note that both DLL test modes have a special set of pin overrides to allow the tester direct control over the DLLs. See Section 17.3

12.5.8.1 DLL low speed test 1 (DLL vendor recommended)

Control Slave Input, Observe Slave Output.

1. set DLL_BYPASS_SLAV= DLL_FORCE_INPUT= 1.

2. hold DLL in reset

3. set slave ADJ[] to max value

4. set TSTCTRL[2:0]=3, TSTCTRL[5:3]=3 (TSTCLK1= slave0_out; TSTCLK2= slave1_out).

5. check that slave output is a buffered version of the slave input. This test can be performed by either applying an oscillating input and observing an oscillating output, or by setting the input to constant values and observing the same values at the output (in our case, this observability is accomplished through the DLL tstclk mux4 by selecting the slave outputs onto TSTCLK1/TSTCLK2 and verifying that the CLK_M90 is present. If we want a constant value on the slave0 input, this can only be accomplished by holding CLK_M90 either high or low, which would also appear to be ok since the DLL is held in reset).

12.5.8.2 DLL low speed test 2 (DLL vendor recommended)

Check Master Through TSTCLK Outputs.
1. set DLL_BYPASS_SLAV= DLL_FORCE_INPUT= 1.
2. hold DLL in reset
3. set MADJ[] to max value
4. set TSTCTRL[2:0]=0, TSTCTRL[5:3]=1 (TSTCLK1= ref_pd; TSTCLK2= fb_pd).
5. check that TSTCLK1 & TSTCLK2 are buffered version of RCLKI.

12.5.9 Fast DDR DLL Test (test mode 23) (whether all DLL tests will be used in mfg. test is still open)

Note the both DLL test modes have a special set of pin overrides to allow the tester direct control over the DLLs. See Section 17.3

12.5.9.1 DLL High Speed Test 1

DLL vendor recommended test:
1. set DLL_BYPASS_SLAV= DLL_FORCE_INPUT= 1.
2. hold DLL in reset.
3. set RCLKI to lowest operating frequency required.
4. set MADJ[] to a nominal value.
5. wait 1us for the analog control to reset
6. release reset, wait 500 RCLKI cycles for the DLL to lock.
7. set TSTCTRL[2:0]=0, TSTCTRL[5:3]=1 (TSTCLK1= ref_pd; TSTCLK2= fb_pd).
8. check that TSTCLK1 & TSTCLK2 have closely aligned rising and falling edges.

12.5.9.2 DLL Functional Slave Test

Recommended by eSilicon:
1. set DLL_BYPASS_SLAV= DLL_FORCE_INPUT= 1
2. hold DLL in reset.
3. set RCLKI to 400MHz.
4. set MADJ[7:0] to 184 (0xb8).
5. wait 1us for the analog control to reset.
6. release reset, wait 500 RCLKI cycles for the DLL to lock.
7. set TSTCTRL[2:0]=3, TSTCTRL[5:3]=3 (TSTCLK1= slave0_out; TSTCLK2= slave1_out)
8. set ASIC pins: DDR_DQSP[8:0]=400MHZ, DDR_DQSN[8:0]=(˜(400MHz)). (this is the input to slave1; slave0_input= CLKM90= RCLKI).
9. set ADJ0[7:0]= 0; ADJ1[7:0]= 0; (slave0_delay= slave1_delay= 562 ps).
10. check the phase relationship of TSTCLK1 & TSTCLK2 relative to RCLKI (i.e. the input clock in step '3' above from the tester). Save this to variables phase_tstclk1_0, phase_tstclk2_0.
11. set ADJ0[7:0]= 92; ADJ1[7:0]= 92; (do not reset the DLL). (slave0_delay= slave1_delay= 1812 ps).
12. check the phase relationship of TSTCLK1 & TSTCLK2 relative to RCLKI. Save this to variables phase_tstclk1_1, phase_tstclk2_1.
13. compare the saved variables:
result0= phase_tstclk1_1 - phase_tstclk1_0;
result1= phase_tstclk2_1 - phase_tstclk2
14. pass/fail: result0 & result1 should both be approx. 1250ps. Note: this test is accomplished on the tester by running one continuous pattern, as follows:
a. apply signals
b. run loop and find measured values 0.
c. break loop.
d. change ADJ[] signals.
e. run loop and find measured values 1.
f. break loop.
g. compare the measured values 0 and 1.
h. pass fail the measured variables.

12.5.10 PCI Functional Tests (test modes 0, 24, 25, or 26)

Loop-back / PRBS tests as described in the PCIe PHY documentation. These can be performed in the normal operating mode of the chip, with all PLLs for non-PCI clocks bypassed (and potentially inactive - test mode 24), with the AB pci_ref_clk PLL bypassed (test mode 25) and, optionally, the Synopsys PCIe PHY PLL bypassed as well (test mode 26).

12.5.11 Fabric Transceiver Functional Test (test modes 27, 28)

Testing and configuration of the Fabric Transceivers is via the ICE9 Serial Control Bus linkage on the SysChain. Most likely, this will be perfomed in a mode (27) which enables the fabric link PLLs in operational mode and drives all other clocks with the test clock input (CCLK, pci_ref_clk, and DCLK PLL is in bypass mode). This test can also be performed in the normal operating mode of the chip or with the sclk PLL bypassed.

For a description of the path to load status into the link control registers see Section **??**, and the link control register descriptions in Section 2.20.

12.6 SysChain

In operation, the ICE9 chip provides a system control scan chain interface (SysChain) to the Module Service Processor (MSP). The MSP uses this chain to load boot code into the ICE9 chip, enable and monitor clocks, assert and release internal reset signals and enable each of the chip's subsystems. The SysChain is also used to read status from the chip and communicate with the processor core EJTAG interfaces. The MIPS EJTAG features are quite powerful and allow almost all of the operations normally obtained with an in-circuit emulator. See the MIPS 5Kf EJTAG specification for further information.

The SysChain functions use the IEEE-JTAG 1149.1 protocol, but the SysChain is not a test feature. It is provided for maintenance and management of the ICE9 chip: JTAG just happens to be a handy protocol to provide this feature. All SysChain chip pins are prefixed with "sch_" and only those pins related to the SysChain carry the "sch_" prefix.

Note that in order to be consistent between the various TAPs, the bit numbering convention for all SysChain TAP registers is MSB closest to TDI, while LSB is closest to TDO.

The SysChain Test Access Port (TAP) consists of eight JTAG controllers wired in series, as shown in Figure 12.7. The first (nearest TDI) is the PCI-Express TAP controller, which has an 8 bit wide Instruction Register (IR). Next is the SysChain TAP controller, which has a 5 bit wide IR. The remaining six controllers are the MIPS EJTAG TAPs, each of which has a 5 bit wide IR. This presents a composite SysChain TAP IR width of $8+5+(6\times5)=43$bits. To complicate matters further, on the ICE9 module the E-Silicon JTAG chain is also wired in series in front of the SysChain, see section 12.6.15 and Figure 12.8. *Therefore the TOTAL Length of the SysTap IR is:*

$$\text{SysTap IR Length: } 18+8+5+(6\times5)=61 \text{ bits.}$$

Note that for all descriptions that follow, the COMPLETE JTAG chain is accounted for. Thus IR length of the System TAP chain includes both the externally (module) wired JTAG as well as the SysChain JTAG.

Each TAP controllers' IR selects which User-defined Data Register (UDR) is connected between that TAP's Test Data Input (TDI) and Test Data Output (TDO) signals. All IR selectable UDR's are documented in section 12.6.5. Note that the relative position of each UDR stays they same, that is, first the selected E-Silicon UDR, followed by the PCI-Express UDR, followed by the ICE9 SysChain UDR, then the six MIPS EJTAG UDRs. Also note that the width each UDR occupies in the chain varies with the UDR selected.

Typically, SysChain accesses will be confined to a UDR in one TAP controller. The MSP will select which TAP and UDR it wishes to access during the initial IR scan, placing the other TAPs into the JTAG BYPASS mode. *When a UDR is being sampled, it is up to software running on the MSP to insure that the proper data values are shifted into this UDR during JTAG Capture-Shift-Update-DR operations to prevent signals from inadvertently changing.*

By wiring the TAP controllers in series there is a small amount of overhead introduced when shifting a particular UDR. Again, referring to Figure 12.7, notice that any E-Silicon UDR has eight downstream TAP controllers that in the best case are in bypass mode. This introduces eight bits of prefix data to any E-Silicon UDR being shifted out. For any PCI-Express UDR the overhead is one bit less and for any SysChain UDR the overhead is two bits

less, since there are only the six MIPS cores downstream of it. When accessing a MIPS Core UDR, the number of overhead bits will vary depending upon which core is being accessed, see Figure 12.7. When shifting data into a UDR the situation is reversed. In either case, the MSP must remember what UDRs have been configured on the chain in order to know their relative positioning.

12.6.1 SysChain Ordering Rules

A write to a syschain register may not immediately take effect, there may be downstream logic that requires extra syschain clocks for the write to complete. If software requires a write to have been completed before doing something else, it must follow the normal system ordering rule, namely read the register back. This read will insure the write has been completed.

12.6.2 Vregs Package

Package

chip_lbs_spec

Attributes

-public_rdwr_accessors

12.6.3 SysChain TAP Constants

Defines

SYSTAP

Constant	Mnemonic	Definition
32'd61	IR_LENGTH	System TAP instruction length
32'd43	SCH_IR_LENGTH	System Chain instruction length
32'd18	JTAG_IR_LENGTH	ESI JTAG TAP controller's instruction length
32'd8	PCI_TAP_IR_LENGTH	PCIe TAP controller's instruction length
32'd5	SCH_TAP_IR_LENGTH	SCH TAP controller's instruction length

12.6.4 SysChain TAP Enumeration

This enumeration allows code to select which TAP is to be operated upon. Software should assume the taps are layed out in the order specified by this enum; see R_SysTapInstrReg for that information as well.

Enum

SysChainTaps

Constant	Mnemonic	Product	Definition
5'h0	ESI		eSilicon TAP
5'h1	PCI		PCI-Express TAP
5'h2	SCH		SysChain TAP
5'h3	CPU2		CPU 2 EJTAG TAP
5'h4	CPU0		CPU 0 EJTAG TAP
5'h5	CPU1		CPU 1 EJTAG TAP
5'h6	CPU3		CPU 3 EJTAG TAP
5'h7	CPU5		CPU 5 EJTAG TAP
5'h8	CPU4		CPU 4 EJTAG TAG

Enum

SysChainTapsTwc

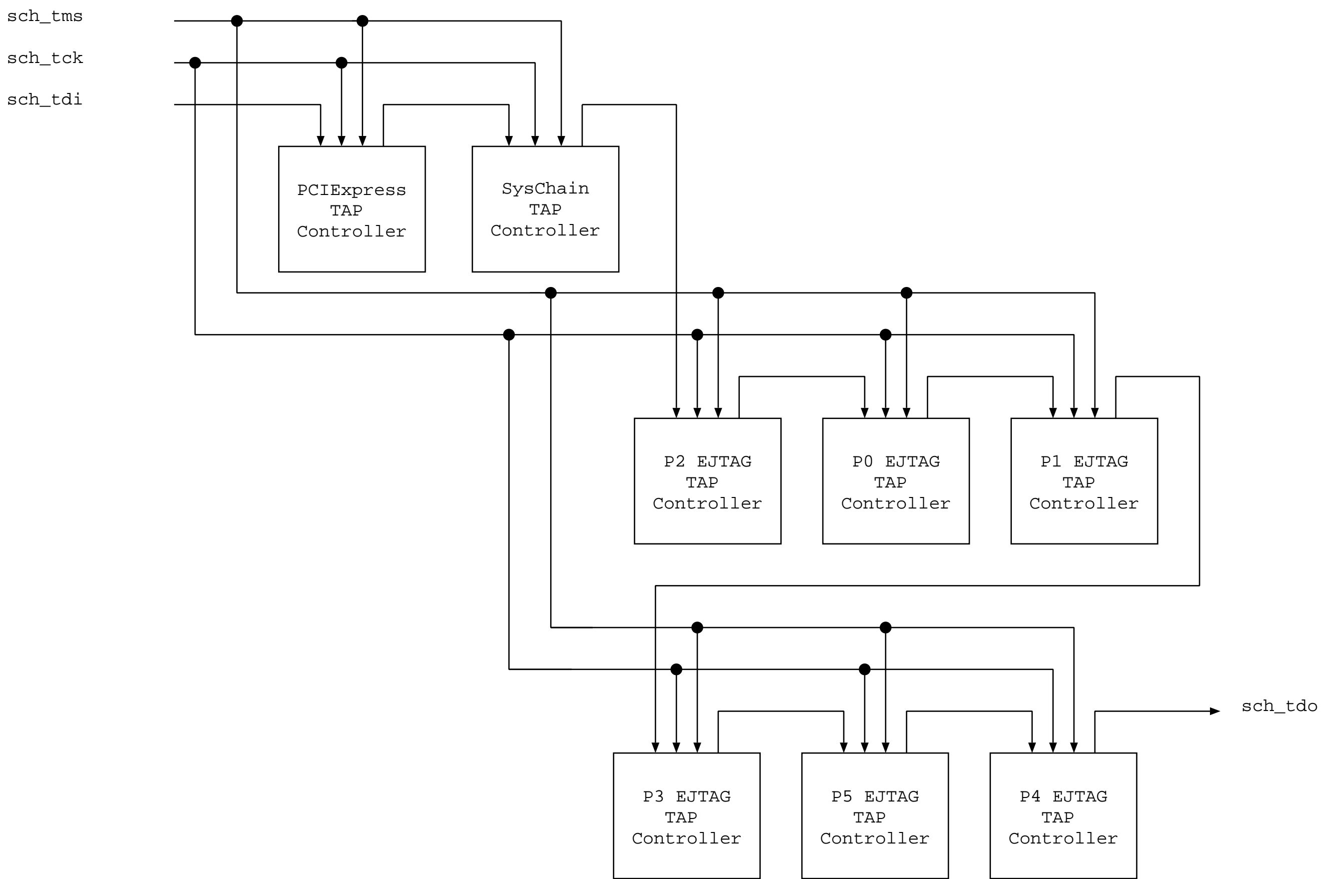


Figure 12.7: SysChain TAP Connections

Constant	Mnemonic	Product	Definition
5'h0	ESI	TWC9A+	eSilicon TAP
5'h1	PCI	TWC9A+	PCI-Express TAP
5'h2	SCH	TWC9A+	SysChain TAP
5'h3	CPU2	TWC9A+	CPU 2 EJTAG TAP
5'h4	CPU0	TWC9A+	CPU 0 EJTAG TAP
5'h5	CPU1	TWC9A+	CPU 1 EJTAG TAP
5'h6	CPU3	TWC9A+	CPU 3 EJTAG TAP
5'h7	CPU5	TWC9A+	CPU 5 EJTAG TAP
5'h8	CPU4	TWC9A+	CPU 4 EJTAG TAG
5'h9	CPU6	TWC9A+	CPU 6 EJTAG TAG
5'ha	CPU7	TWC9A+	CPU 7 EJTAG TAG
5'hb	CPU8	TWC9A+	CPU 8 EJTAG TAG
5'hc	CPU9	TWC9A+	CPU 9 EJTAG TAG

note: twc9 order TBD

12.6.5 System TAP Instructions

Description

The System TAP instruction enumerations can be loaded into their respective JTAG TAP Controller IRs to select any one of the UDRs listed. Each UDR is documented further in the sections that follow. There is one set of enumerations per TAP Controller. Only the ICE9's SysChain TAP enumerations are fully described in this spec. The remaining TAPs are fully documented in their respective specifications.[2]

The SysTapEsiInstr enumeration below is a special case. The E-Silicon TAP IR is only 18 bits wide, but the enumerations are specified as 26 bits wide to accommodate unique enumerations for DR's of different sizes using a single IR encoding. This is needed because the E-Silicon TAP supports an IEEE P1500 TAP controller as one of the devices that can be connected to its scan chain. The P1500 can connect DRs of different sizes depending upon what was written to the JPC or SMS IR, even though in each case the E-Silicon IR TAP encoding is the same. Thus the P1500 breaks the typical one-to-one correlation between the E-Silicon TAP IR selected and the associated DR length. In order to avoid maintaining state information in software to deal with the P1500; the enumerations in this table were widened to allow software to specify directly the context of which JPC or SMS WDR is being selected during the current scan operation. Note that in every case the least significant 18 bits of the encodings are identical. This is what is shifted into the E-Silicon TAP IR. The remaining 8 bits are not scanned into the TAP, but used by software to indicate the length of the associated DR register.

For the ICE9, there are important deviations from the JTAG Standard within the E-Silicon TAP. The E-Silicon TAP uses an inverted TCK internally. When connected to JTAG scan chains that do not use the inverted TCK, this has the side-effect of inducing one extra clock of delay to the shift chain across the E-Silicon TAP. Therefore shifting data into or out of scan registers within the E-Silicon TAP require one extra TCK be inserted ahead of the shift. In the special case of *reading* the SMS 512K Test Algo. or Status Registers the E-Silicon TAP requires *two* extra TCKs be inserted prior to shifting data out of the register.

In addition, all JPC and SMS WDR registers shift in a direction opposite of the normal IEEE JTAG standard, having their MSB connected to TDO and LSB connected to TDI instead of the other way round. Thus the contents of these registers may need to be bit-swapped, depending upon how a given JTAG bus master shifts its scan chain.

Enum

SysTapEsiInstr

Constant	Mnemonic	Definition	(TapSize)	(Capture?)	(Update?)
26'h00_00000	BYPASS0	Bypass	1	N	N
26'h00_3FFFE	IDECODE	Device Identification Register*	32	Y	N
26'h00_3FFFF	BYPASS	Bypass	1	N	N

[2]For the E-Silicon JTAG TAP see section <tbd> entitled <tbd> in the document *<tbd>*. For the PCI-Express JTAG TAP see Section 7.2 entitled "JTAG Interface" in the document, *PCIe1^{TM}90mm PHY Databook*. For the MIPS EJTAG TAP see Chapter 10 entitled "EJTAG Debug Feature" in the *MIPS64^{TM}5K^{TM}Processor Core Family Software User's Manual*.

Constant	Mnemonic	Definition	(TapSize)	(Capture?)	(Update?)
26'h00_3FC7A	SELECT_JPC_WIR	Select JPC_WIR	6	N	Y
26'h00_3FD7A	SELECT_JPC_WDR	Select JPC_WDR	5	Y	Y
26'h01_3FD7A	SELECT_JPC_WDR_SMSNUM	Select JPC SMS Num Register	5	Y	Y
26'h02_3FD7A	SELECT_JPC_WDR_BYPASS	Select JPC Bypass Register	1	N	N
26'h00_3FE7A	SELECT_SMS_WIR	Select SMS_WIR	6	N	Y
26'h00_3FF7A	SELECT_SMS_WDR	Select SMS_WDR	6	Y	Y
26'h01_3FF7A	SELECT_SMS_WDR_TBX32K	Select SMS 32K Test Algo. Reg	234	Y	Y
26'h02_3FF7A	SELECT_SMS_WDR_TBX512K2P	Select SMS 512K2P Test Algo. Reg	556	Y	Y
26'h03_3FF7A	SELECT_SMS_WDR_TBX512K1P	Select SMS 512K1P Test Algo. Reg	308	Y	Y
26'h04_3FF7A	SELECT_SMS_WDR_STS32K	Select SMS 32K Status Reg.	6	Y	N
26'h05_3FF7A	SELECT_SMS_WDR_STS512K	Select SMS 512K Status Reg.	6	Y	N
26'h06_3FF7A	SELECT_SMS_WDR_BYPASS	Select SMS Bypass	1	N	N
26'h00_3FFE8	EXTEST	Extest	1 (?TBD)		
26'h00_3FFF8	SAMPLE	Sample	1 (?TBD)		
26'h00_3FFF8	PRELOAD	Preload (same value as Sample)	1 (?TBD)		
26'h00_3FFCF	HIGHZ	Highz	1 (?TBD)		
26'h00_3FFEF	CLAMP	Clamp	1 (?TBD)		

<TBD - Add the remaining E-Silicon TAP Instructions>
* = Test-Logic-Reset Default

Enum

JpcSms

Attributes

-descfunc

Constant	Mnemonic	Product	Definition
5'h1	BBS		chip.bbs
5'h2	CAC0		chip.ps0.cac
5'h3	CAC1		chip.ps1.cac
5'h4	CAC2		chip.ps2.cac
5'h5	CAC3		chip.ps3.cac
5'h6	CAC4		chip.ps4.cac
5'h7	CAC5		chip.ps5.cac
5'h8	COHO		chip.coho
5'h9	COHE		chip.cohe
5'ha	CPU0		chip.ps0.cpu.m5kf
5'hb	CPU1		chip.ps1.cpu.m5kf
5'hc	CPU2		chip.ps2.cpu.m5kf
5'hd	CPU3		chip.ps3.cpu.m5kf
5'he	CPU4		chip.ps4.cpu.m5kf
5'hf	CPU5		chip.ps5.cpu.m5kf
5'h10	DDRE		chip.ddre.ddi
5'h11	DDRO		chip.ddro.ddi
5'h12	DMA		chip.dma
5'h13	FSW		chip.fsw

Enum

SysTapPciInstr

Constant	Mnemonic	Definition	(TapSize)	(Capture?)	(Update?)
8'h01	IDECODE	Device Identification Register*	32	Y	N

Constant	Mnemonic	Definition	(TapSize)	(Capture?)	(Update?)
8'h0D	USERCODE	User Code Register	32	Y	N
8'h31	CRSEL	Control Register	18	Y	Y
8'h3D	APUCRSEL	APU Control Register	18	Y	Y
8'hA1	OVRDREG	OVRD Register	45	Y	Y
8'hAD	EXTEST	Extest	1(?TBD)		
8'hC1	EXTEST_TRAIN	Extest training	1(?TBD)		
8'hCD	EXTEST_PULSE	Extest pulse	1(?TBD)		
8'hF1	PRELOAD	Preload	1(?TBD)		
8'hF1	SAMPLE	Sample	1(?TBD)		
8'hFF	BYPASS	Bypass (all unused codes are bypass)	1		

* = Test-Logic-Reset Default

Enum

SysTapSchInstr

Constant	Mnemonic	Product	(RegName)	Definition	(TapSize)	(Capture?)	(Up
5'h00	BYPASS0			Bypass 0	1		
5'h01	IDECODE		R_SysTapIDecode	Device Identification Register*	32	Y	
5'h08	PLL		R_SysTapPll	PLL Control Register	64	Y	
5'h09	RESET		R_SysTapReset	Reset Control Register	64	Y	
5'h0A	CPUDINT	ICE9A	R_SysTapDint	CPU Debug Interrupt Control Register	8	Y	
5'h0B	SMSBIST		R_SysTapSmsBist	SMS RAM BIST Control Register	16	Y	
5'h0C	SCB		R_SysTapScb	Serial Configuration Bus Interface Register	64	Y	
5'h0D	ATNMSP		R_SysTapAtnMsp	Attention MSP Register	32	Y	
5'h0F	MEMINIT	TWC9A	R_SysTapMemInit	Memory Zero Register	32	Y	
5'h10	SCB64	TWC9A	R_SysTapScb64	Serial Configuration Bus 64-bit access Register	104	Y	
5'h1F	BYPASS			Bypass	1		

* = Test-Logic-Reset Default

Enum

SysTapCpuInstr

Constant	Mnemonic	Definition	(TapSize)	(Capture?)	(Update?)
5'h01	IDECODE	Device Identification Register*	32	Y	N
5'h03	IMPCODE	Implementation Register	32	Y	N
5'h08	ADDRESS	Address Register	36	Y	Y
5'h09	DATA	Data Register	64	Y	Y
5'h0A	CONTROL	EJTAG Control Register	32	Y	Y
5'h0B	ALL	Address, Data and EJTAG Control Registers	132	Y	Y
5'h0C	EJTAGBOOT	Forces Debug Exception after Reset.	1		
5'h0D	NORMBOOT	Execute reset handler after Reset.	1		
5'h0E	FASTDATA	Data and Fastdata Registers	65	Y	Y
5'h1F	BYPASS	Bypass	1		

* = Test-Logic-Reset Default

12.6.6 System TAP Instruction Register

Description

The System Test Access Port Instruction Register consists of the all JTAG TAP IRs concatenated together. This is used only in ICE9, for TWC9 see R_SysTapInst.

Class

R_SysTapInstrReg

Attributes

-tapSize=61

Bit	Mnemonic	Access	Reset	Product	Definition
60:43	Esi	W	1		E-Silicon TAP Instruction Register
42:35	Pci	W	1		PCI Express TAP Instruction Register
34:30	Sch	W	1		System Chain TAP Instruction Register
29:25	Cpu2	W	1		CPU 2 TAP Instruction Register
24:20	Cpu0	W	1		CPU 0 TAP Instruction Register
19:15	Cpu1	W	1		CPU 1 TAP Instruction Register
14:10	Cpu3	W	1		CPU 3 TAP Instruction Register
9:5	Cpu5	W	1		CPU 5 TAP Instruction Register
4:0	Cpu4	W	1		CPU 4 TAP Instruction Register

12.6.7 System TAP Instruction Register for TWC9

Description

The System Test Access Port Instruction Register consists of the all JTAG TAP IRs concatenated together. This is used only in TWC9, for ICE9 see R_SysTapInst.

Class

R_SysTapInstrTwc

Attributes

-tapSize=81

Bit	Mnemonic	Access	Reset	Product	Definition
80:63	Esi	W	1	TWC9A+	E-Silicon TAP Instruction Register
62:55	Pci	W	1	TWC9A+	PCI Express TAP Instruction Register
54:50	Sch	W	1	TWC9A+	System Chain TAP Instruction Register
49:45	Cpu2	W	1	TWC9A+	CPU 2 TAP Instruction Register
44:40	Cpu0	W	1	TWC9A+	CPU 0 TAP Instruction Register
39:35	Cpu1	W	1	TWC9A+	CPU 1 TAP Instruction Register
34:30	Cpu3	W	1	TWC9A+	CPU 3 TAP Instruction Register
29:25	Cpu5	W	1	TWC9A+	CPU 5 TAP Instruction Register
24:20	Cpu4	W	1	TWC9A+	CPU 4 TAP Instruction Register
19:15	Cpu7	W	1	TWC9A+	CPU 7 TAP Instruction Register
14:10	Cpu6	W	1	TWC9A+	CPU 6 TAP Instruction Register
9:5	Cpu9	W	1	TWC9A+	CPU 9 TAP Instruction Register
4:0	Cpu8	W	1	TWC9A+	CPU 8 TAP Instruction Register

12.6.8 Device Identification Register

Description

The Device Identification (IDECODE) Register contains the ICE9 and Sicortex device specific information in the IEEE 1149.1 JTAG Standard format.

Class

R_SysTapIDecode

Attributes

-tapSize=32

Bit	Mnemonic	Access	Reset	Type	Definition
31:28	Version	R	pins		Sicortex part version for the ICE9 device. Returns 1 for ICE9A0/ICE9B0, 2 for ICE9A1/B1, etc.
27:12	PartNumber	R	pins	AddrProduct	Sicortex part number for the ICE9 device. Always ICE9.
11:1	ManufId	R	SICORTEX	AddrTapMfgr	JEDEC derived IEEE 1149.1 manufacturer identifier for SiCortex
0	JtagOne	R	0x1		IEEE 1149.1 JTAG required constant '1'

12.6.9 PLL Control Register

Description

The PLL Control Register chain has one control and status register for each PLL on the ICE9. The registers control the input signals described in Tables 12.2 and 12.3. The PLL Control Register chain also has a 3-bit register for each of the 2 PLL groups (Pllsw & Pllne) that makes one of the clocks in the group observable through pins on the chip. The order of the bits in the scan chain across the five PLLs and the two clock output control registers is shown in the attribute table below. The reset values should be such that the PLLs run at their nominal system speeds, to minimize the complexity of the ATE initialization sequence.

Class

R_SysTapPll

Attributes

-tapSize=64

Bit	Mnemonic	Access	Reset	Type	Definition
63					Reserved
62	IclkReset	RW	1		PCI PHY and PMI Clock PLL Reset
61	IclkLock	R	0		PMI Clock PLL Lock (1=locked, 0=unlocked)
60:58	Pllsw	RW	0		Clock output control register (see *Pllsw* description below).
57:55	Pllne	RW	0		Clock output control register (see *Pllne* description below).
54	D1clkReset	RW	0		DDR1 Controller Clock PLL Reset.
53:49	D1clkDivf	RW	23		DDR1 Controller Clock PLL Divisor Factor.
48:47	D1clkOutSel	RW	1		DDR1 Controller Clock PLL Output Select.
46	D1clkBypClkSel	RW	0		DDR1 Controller Clock PLL Bypass Clock Select
45	D1clkBypEnb	RW	0		DDR1 Controller Clock PLL Bypass Enable.
44	D1clkLock	R	0		DDR1 Controller Clock PLL Lock (1=locked, 0=unlocked).
43	D0clkReset	RW	0		DDR0 Controller Clock PLL Reset.
42:38	D0clkDivf	RW	23		DDR0 Controller Clock PLL Divisor Factor.
37:36	D0clkOutSel	RW	1		DDR0 Controller Clock PLL Output Select.
35	D0clkBypClkSel	RW	0		DDR0 Controller Clock PLL Bypass Clock Select
34	D0clkBypEnb	RW	0		DDR0 Controller Clock PLL Bypass Enable.

Bit	Mnemonic	Access	Reset	Type	Definition
33	D0clkLock	R	0		DDR0 Controller Clock PLL Lock (1=locked, 0=unlocked).
32	PciRefReset	RW	0		PCI Reference Clock PLL Reset.
31:27	PciRefDivf	RW	11		PCI Reference Clock PLL Divisor Factor.
26:25	PciRefOutSel	RW	2		PCI Reference Clock PLL Output Select.
24	PciRefBypDiv2Enb	RW	0		PCI Reference Clock Bypass Divide by 2 Enable.
23	PciRefBypEnb	RW	0		PCI Reference Clock Bypass Enable.
22	PciRefLock	R	0		PCI Reference Clock PLL Lock (1=locked, 0=unlocked).
21	SclkReset	RW	0		Switch Fabric SERDES Clock PLL Reset.
20:16	SclkDivf	RW	11		Fabric Switch and Links Clock PLL Divisor Factor.
15:14	SclkOutSel	RW	0		Fabric Switch and Links Clock PLL Output Select.
13	SclkBypDiv2Enb	RW	0		Fabric Switch and Links Clock PLL Bypass Divide by 2 Enable.
12	SclkBypEnb	RW	0		Fabric Switch and Links Clock PLL Bypass Enable.
11	SclkLock	R	0		Fabric Switch and Links Clock PLL Lock (1=locked, 0=unlocked).
10	PclkReset	RW	0		Processor Clock PLL Reset Reset.
9:5	PclkDivf	RW	14		Processor Clock PLL Divisor Factor.
4:3	PclkOutSel	RW	0		Processor Clock PLL Output Select.
2	PclkBypDiv2Enb	RW	0		Processor Clock PLL Bypass Divide by 2 Enable.
1	PclkBypEnb	RW	0		Processor Clock PLL Bypass Enable.
0	PclkLock	R	0		Processor Clock PLL Lock (1=locked, 0=unlocked).

The PLL control register chain also has a 3-bit register for each of the 2 PLL groups (Pllsw & Pllne) that makes one of the clocks in the group observable through pins (test_clk_o_h/l in Pllsw or pci_ref_clk_h/l for Pllne) See Table 12.13 below for a complete description of which clocks are made observable for each of these registers based on the settings of these two registers.

For both Pllsw and Pllne, the reset default value causes the differential outputs to be tri-stated. With an operating PCIe interface, an ICE9 would need to have the Pllne register set to select pci_ref_clk. For the ddr-clock PLLs, we also allow for driving out the XOR of the in-phase and 90 degree phase shifted PLL outputs. This allows for a crude measure of phase alignment of the 2 clocks; if they're exactly 90 degrees out of phase, the XOR signal will have a 50% duty cycle. Since we won't use a real analog mixer for the XOR, the resulting signal will be only a rough approximation to the ideal.

In all cases, what's driven to the output mux & LVDS output cell is taken from very close to the PLL output, i.e., near the root of the clock tree, not tapped off the end of the clock tree. The provided functionality is for testing PLL operation, not the clock distribution network.

Bit Field	*Values*	*Read/Write*	*Value after Reset*	*Description (**Pllsw**)*	*Description (**Pllne**)*
<2:0>	0	RW	0	select no output (HiZ)	select no output (HiZ)
	1			select sys_clk_o	select pci_ref_clk
	2			select no output (HiZ)	select iclk (from PCIe PHY)
	3			select sclk	select pclk
	4			select sclk_x2	select cclk (pclk_div2)
	5			select d1clk	select d0clk
	6			select d1clk90	select d0clk90
	7			select (d1clk .XOR. d1clk90)	select (d0clk .XOR. d0clk90)

Table 12.13: Clock Output Control Register (2 copies)

12.6.10 Reset Control Register

Description

The Reset Control Register allows the MSP to assert resets and enables on a unit by unit basis. All reset signals are SET after a hardware reset. All enables are CLEAR after a hardware reset. All reset and enable bits are directly read upon a SysChain *Capture-DR* operation and directly written on an *Update-DR*.

There are two types of reset implemented by the Reset Control Register; Unit resets and Virage STAR Memory System (SMS) resets. The Unit resets are used to initialize specific functional units within the ICE9. The SMS resets are used to reset the Built In Self Test (BIST) status for all the SMS RAMs in the ICE9.

At power-on both Unit and SMS resets are asserted. The MSP will bring the ICE9 out of reset by first de-asserting the SMS resets so that BIST can be performed on all RAMs that support it while keeping the Unit resets asserted. In the ICE9, BIST is used not only for testing RAM but also to initialize some RAMs into a useful state for system bring-up. During BIST, it is necessary that each functional unit that contains SMS RAM be held in reset, to prevent improper operations from being induced by the BIST activities. Once BIST has successfully completed, the MSP will bring the functional units out of reset by de-asserting the appropriate Unit reset bits as part of system bring-up.

Restrictions

Whenever the MSP is changing more than one of these bits in a single *Update-DR* operation, it *must not* set bits while clearing others. *All multi-bit operations must be isotonic (all set or all clear).* This restriction avoids race hazards in downstream logic that may use combinatorial expressions made from more than one of these bits.

Class

R_SysTapReset

Attributes

-tapSize=64

Bit	Mnemonic	Access	Reset	Product	Definition
63:50					Reserved
49	Lac	RW	1	TWC9A+	LAC reset. Prior to TWC9, this was ganged into the Scbm reset.
48	Pmi	RW	1	TWC9A+	PMI reset. Prior to TWC9, this was ganged into the Scbm reset.
47:44	ProcSms6	RW	0x3F	TWC9A+	Reset for Processor 9:6 SMS (Pclk). See _ProcSms6.
(**)	(**)				Reserved. FIX; spread ProcSms6 and Proc6 to allow room for CPU10-15.
43:40	Proc6	RW	0x3F	TWC9A+	Reset for Processor 9:6. See _Proc.
39	SmsClkEnb	RW	1		SMS Clock Enable
38	Ddr0Sms	RW	1		Reset for DDR0 controller SMS (D0clk).
37	Ddr1Sms	RW	1		Reset for DDR1 controller SMS (D1clk).
36	CoheSms	RW	1		Reset for COH and DDI even SMS (Cclk).
35	CohoSms	RW	1		Reset for COH and DDI odd SMS (Cclk).
34	FabSwSms	RW	1		Reset for Fabric Switch SMS (Sclk).
33	DmaSms	RW	1		Reset for DMA Engine SMS (Cclk).
32	CswOclaSms	RW	1		Reset for Central Switch OCLA SMS (Cclk).
31	L2CacSms	RW	1		Reset for Level 2 Cache SMS (Cclk).
30	ScbmSms	RW	1		Reset for SCBM SMS (Cclk).
29	BbsSms	RW	1		Reset for BBS SMS (Cclk).
28	PciSms	RW	1		Reset for PCI SMS (Iclk).
27:22	ProcSms	RW	0x3F		Reset for Processor 5:0 SMS (Pclk) [six resets, one per SMS]. See also _ProcSms6.
21	Dimm0	RW	1		Reset for DIMM0.
20	Dimm1	RW	1		Reset for DIMM1.
19	Ddr0	RW	1		Reset for DDR0 controller.
18	Ddr1	RW	1		Reset for DDR1 controller.
17	Cohe	RW	1		Reset for COH and DDI even.
16	Coho	RW	1		Reset for COH and DDI odd.
15	FabSw	RW	1		Reset for fabric switch.
14	FabLn	RW	1		Reset for fabric links.

Bit	Mnemonic	Access	Reset	Product	Definition
13	Dma	RW	1		Reset for DMA engine.
12	Csw	RW	1		Reset for Central Switch.
11	L2Cac	RW	1		Reset for Level 2 Caches.
10	Scb	RW	1		Reset for SCB. Prior to TWC9A, this also reset the BBS including the OCLA.
9	I2c	RW	1		Reset for I2C.
8	UartIoEnb	RW	0		Enable for UART I/O.
7	Uart	RW	1		Reset for UART.
6	Pci	RW	1		Reset for PCI.
5:0	Proc	RW	0x3F		Reset for Processor 5:0 [six resets, one per processor]. See also _Proc6. This will reset all processor registers, excluding the R_IcetxTime register.

12.6.11 Memory Init Register

Description

The Memory Init Register allows the MSP to initalize on chip memories on a unit by unit basis. Memories are NOT reset by default and the MSP must use this register to insure proper memory state.

Class

R_SysTapMemInit

Attributes

-tapSize=32

Bit	Mnemonic	Access	Reset	Product	Definition
31:26				TWC9A+	Reserved. (For extending CPUs to 15:10)
25:16	Cpu	RW	0	TWC9A+	Init Processor 9:0. One per processor.
15:13				TWC9A+	Reserved.
12	Lac	RW	0	TWC9A+	Init LAC.
11	Pmi	RW	0	TWC9A+	Init PMI.
10	Ddr	RW	0	TWC9A+	Init DDR0 + 1 controller (D1clk).
9	Cohe	RW	0	TWC9A+	Init COH and DDI even (Cclk).
8	Coho	RW	0	TWC9A+	Init COH and DDI odd (Cclk).
7	Fabsw	RW	0	TWC9A+	Init Fabric Switch (Sclk).
6	Dma	RW	0	TWC9A+	Init DMA Engine (Cclk).
5	CswOcla	RW	0	TWC9A+	Init Central Switch OCLA (Cclk).
4	L2Cac	RW	0	TWC9A+	Init Level 2 Cache (Cclk).
3	Scbm	RW	0	TWC9A+	Init SCBM (Cclk).
2	Bbs	RW	0	TWC9A+	Init BBS (Cclk).
1	Pci	RW	0	TWC9A+	Init PCI (Iclk).
0	Done	R	0	TWC9A+	Init busy. To initalize a memory, software writes the appropriate bits one. This bit will then remain cleared until all RAMs are finished, at which point it will read as a one. Software must then zero this register. Once the register is zero, the MSP has the option of initalizing other memories.

12.6.12 Processor Debug Interrupt Register

Description

The Processor Debug Interrupt Control Register allows the MSP to send a Debug Interrupt (DINT) request to one or more MIPS cores in the ICE9. The MIPS EJTAG Specification specifies that a debug interrupt is requested when the DINT signal transitions from low to high.[3] The associated MIPS core is allowed to synchronize this signal to its own clock before detecting its rising edge. Section 8.2.2 of the specification also states that the DINT high and low times must observe a minimum of 1uS in order to leave enough time for the CPU core to synchronize the DINT signal to its internal clock domains. The DINT signal rise/fall times are also specified for a maximum of 3nS. The MSP and associated logic should observe these restrictions for bits in this register.

This register only exists in ICE9A. In ICE9B it was moved to R_ScbDInt.

Class

R_SysTapDint

Attributes

-tapSize=8

Bit	Mnemonic	Access	Reset	Product	Definition
7				ICE9A	Reserved.
6	CpuDintEnb	RW	0	ICE9A	Enable any processor or OCLA to send a debug interrupt to all processors.
5	Dint5	RW	0	ICE9A	Processor Core 5 Debug Interrupt (on transition from 0 to 1).
4	Dint4	RW	0	ICE9A	Processor Core 4 Debug Interrupt (on transition from 0 to 1).
3	Dint3	RW	0	ICE9A	Processor Core 3 Debug Interrupt (on transition from 0 to 1).
2	Dint2	RW	0	ICE9A	Processor Core 2 Debug Interrupt (on transition from 0 to 1).
1	Dint1	RW	0	ICE9A	Processor Core 1 Debug Interrupt (on transition from 0 to 1).
0	Dint0	RW	0	ICE9A	Processor Core 0 Debug Interrupt (on transition from 0 to 1).

12.6.13 SMS BIST Contol Register

Description

The SMS BIST Control Register allows the MSP to initiate BIST on all of the Virage SMS RAMs inside the ICE9. To insure proper operation, BIST should only be initiated after every SMS reset has been de-asserted in the Reset Control Register. SMS BIST performs RAM tests, loads the memory fuse map and performs initialization on those RAMs that require specific data initialization prior to normal operation. This is important for Tag arrays and some other memory structures that, because of the BIST requirement, can't be initialized under reset. BIST is activated via the Virage SMART signals; which are entirely separate from the P1500 port connected to the test JTAG chains. The attribute table below shows the format of the register.

For chips installed in systems, all Virage BIST operations are completed while unit resets are asserted, see 12.6.10. This includes the INITIALIZE operation. The proper behavior for all components on the chip that have RAM arrays is to clear all address registers to 0 while RESET is asserted and the RAM is not in INITIALIZE mode. While RESET is asserted and the RAM is in INITIALIZE mode, the hardware should clear all locations to a known and repeatable state. INITIALIZE and BITS commands should be ignored when RESET is not asserted.

The MSP prepares for Virage BIST by first clearing all of the SMS Resets in the Reset Control Register, making sure that the Unit resets remain asserted to prevent unpredictable hardware operations while BIST is running. The MSP then enables Virage BIST by asserting both SmartEnb and SmartRun bits in R_SysTapSmsBist and then de-asserting SmartEnb. BIST is complete for all SMS RAMs when the SmartDone bit is asserted. The MSP must poll this bit to determine when BIST has completed. After BIST completion, the MSP can examine the SmartFail bit to determine if BIST passed or failed. The MSP should be aware that one of the SMART failure modes is the inability to complete and should timeout after a suitable polling period has elapsed and SmartDone has not asserted.

[3]"EJTAG Specification", Revision 3.10, MIPS Technologies document number MD00047.

SMART activity can be altered by writing to the other control bits in this register prior to setting the SmartRun bit. For normal system bring-up the MSP should leave the other writable bits at their reset defaults. This allows SMART testing to load the hardware programmed repair mask before running BIST across all SMS groups. De-asserting the SMS CLK Enable bit in the Reset Control Register will inhibit BIST operation.

Class

R_SysTapSmsBist

Attributes

-tapSize=16

Bit	Mnemonic	Access	Reset	Type	Definition
15:9					Reserved
8	SmartDone	R	0		SMART Done (0=not-done, 1=done).
7	SmartFail	R	0		SMART Failure (only valid after SmartDone bit is set; 0=passed, 1=failed).
6	SmartReady	R	1		BIST Group Done (signals when the current SMS BIST group has finished).
5	CurrentError	R	0		BIST Group Failure (signals when the current SMS BIST group has failed).
4	RunBist	RW	1		Run BIST as part of SMART testing.
3	HardRepair	RW	1		Use hardware programmed repair mask (enable before BIST).
2	SoftRepair	RW	0		Use software programmed repair mask (leave disabled).
1	SmartEnb	RW	0		Enable SMART testing.
0	SmartRun	RW	0		Runs SMART on transition to '1', clears SmartDone on transition to '0'.

12.6.14 Serial Configuration Bus Interface Register

Description

The Serial Configuration Bus (SCB) Interface Register allows the MSP to communicate with devices on the SCB. All chip clocks need to be running when the SysChain accesses the SCB. In chip test mode we ensure this by putting all but the fabric SERDES clocks in bypass mode. In a system we ensure this by either tying all clocks into bypass mode to SCH_TCK or by starting all the PLLs.

Any register on the SCB may be written from the SysChain. The SCB mechanism is particularly useful in testing the fabric link hardware. The attribute table shows the layout of the SCB scan register. There is just one SCB scan register on the ICE9 chip.

Class

R_SysTapScb

Attributes

-tapSize=64

Bit	Mnemonic	Access	Reset	Type	Definition
63:32	Data	RW	0		Read/Write Data. On writes, data to be written. On reads, when _Busy is cleared, the read data.
31	Reset	RW	0		Reset SCB slaves. Applied when _Go set. On the next “Go”, before sending the read or write transaction, first send a RESET. This is a method of last resort - one short of asserting a real reset wire - to allow hung slaves to be accessed.
30:2	Addr	RW	0		Address. Applied when _Go set.
1	Write	W	0		Write, not read. Applied when _Go set. Assert for writes, clear for reads.
1	Busy	R	0		Command busy. SCB sets this return on a “Go” and clears it when a SysChain write completes or a read returns data. Overlaps allowed.
0	Go	W	0		Go and start command. Must be a one for the SCB to process this command. The SCB will then clear this bit in the response.

Class

R_SysTapScb64

Attributes

-tapSize=104

Bit	Mnemonic	Access	Reset	Product	Definition
103:99				TWC9A+	Reserved
98	Reset	RW	0	TWC9A+	Reset SCB slaves. Applied when _Go set. On the next “Go”, before sending the read or write transaction, first send a RESET. This is a method of last resort - one short of asserting a real reset wire - to allow hung slaves to be accessed.
97	Busy	R	0	TWC9A+	Command busy. SCB sets this return on a “Go” and clears it when a SysChain write completes or a read returns data. Note the R_SysTapScb register has this bit overlapped with _Go, here it is separate.
96	Go	W	0	TWC9A+	Go and start command. Must be a one for the SCB to process this command. The SCB will then clear this bit in the response.
95				TWC9A+	Reserved.
94:66	Addr	RW	0	TWC9A+	Address bits 30:2. Saved when _Go set.
65	Write	RW	0	TWC9A+	Write, not read. Applied when _Go set. Assert for writes, clear for reads.
64	Dword	RW		TWC9A+	Doubleword access. Applied when _Go set. Indicates this transaction is 64 bits instead of 32 bits. Note 32 bit transactions write and return data in naturally aligned position, that is if _Addr[2] is set, then _Data[63:32] is used.
63:0	Data	RW	0	TWC9A+	Read/Write Data. On writes, data to be written. On reads, when _Busy is cleared, the read data.

12.6.15 MSP-Hosted Node Attention Register

R_SysTapAtnMsp provides the MSP manipulated side of the MSP to node chip communication channel. When used in conjunction with the node chip manipulated register R_ScbAtnChip, two-way communication can be provided via the SysChain between software running on the MSP and software running on the node chip. See 10.14.12 for a more detailed description of the R_ScbAtnChip register.

To send a 25-bit character to the chip, the MSP polls until SendVld is clear. The MSP then writes SendData and writes a one to SendVld. To receive a 25-bit character, the MSP polls for RecvVld set, reads the data from RecvData and then writes a one to RecvTaken.

Note that a register read or 8 SysChain clocks must occur after any write to this register for the write to take effect (see 12.6.1).

Class

R_SysTapAtnMsp

Attributes

-tapSize=32

Bit	Mnemonic	Access	Reset	Product	Definition
31:30				ICE9A	Reserved.
31	SendReq	W1CS	0	ICE9B+	Send Data Request. Write one to set and indicate new send data for chip. This will cause _SendVld to assert.
30	TxAtnMask	RW	0	ICE9B+	Transmit Attention Mask. Write one to indicate sys_atn_l pin should be asserted if _SendVld is clear, indicating new data may be sent. If clear, sys_atn_l is not asserted for this reason. Note that _SendVld is clear in the idle steady state, so to prevent permanent attention this bit should be cleared when there is no data to be sent. Overlaps Allowed.
29	NonComAtn	R	0	ICE9B+	Non-Communication Attention Request. Attention is required for other then AtnMsp register reasons. A duplicate of the R_ScbAtnInt_NonComAtn bit to avoid the MSP having to change instruction registers in the fast path. (Note writing this bit has no effect, so old ICE9A code that writes _RecvAtn will NOP.)
29	RecvAtn	RW	0	ICE9A	Receive Attention Enable. Write one to indicate sys_atn_l pin should be asserted if _RecvVld is also asserted. If clear, sys_atn_l is never asserted for this reason. Overlaps Allowed.
28	RecvTaken	W1C	0		Receive Data Taken. Write one to send to chip indication that RecvData was accepted, and clear _RecvVld.
27	RecvVld	R	0		Receive Data Valid. Valid flag from Chip, one indicates Data contains new receive data. Cleared by writing one to _RecvTaken.
26	SendVld	RW1CS(*)	0	(See Text)	Send Data Valid. ICE9A: RW1S; write one to set and indicate new send data for chip. ICE9B+: Read only, write using _SendReq instead. BOTH: Read to indicate send data pending for chip. Cleared when chip takes the data.
25:0	RecvData	R	0		Receive Data. Overlaps SendData. If RecvVld is set, returns the next data to be received from the MSP. Note this is different data then that written.
25:0	SendData	W	0		Send Data. Overlaps RecvData. If SendVld is simultaniously being written with a one, enqueues new send data for the chip, and sets SendVld. If SendVld is not being written with a one, this is ignored. This enables the MSP on a read of this register to set only bit 29 inbound, and not recirculate other bits.

12.6.16 External JTAG Chains

Figure 12.8 shows how the SysChain and JTAG TAPs are connected on the CPU module. All nine ICE9 TAPs are connected in series, and share common TRST, TCK, and TMS lines. TDI, TRST, and TCK are distributed module-wide; all ICE9s see the same values of these signals at all times. TMS is separately distributed to each ICE9 to facilitate manipulating a subset of the ICE9s on a module without having to place the others in reset or bypass mode. TDO is individually multiplexed from each ICE9 to allow the MSP to receive a single ICE9's serial data even if multiple ICE9s are being scanned.

12.7 Global reset

The ICE9 chip implements a 2-level reset strategy. Hard-reset (normally asserted at power-on) is a chip pin. To provide for reset of parts of the chip under module-service-processor control there are soft-reset bits from the

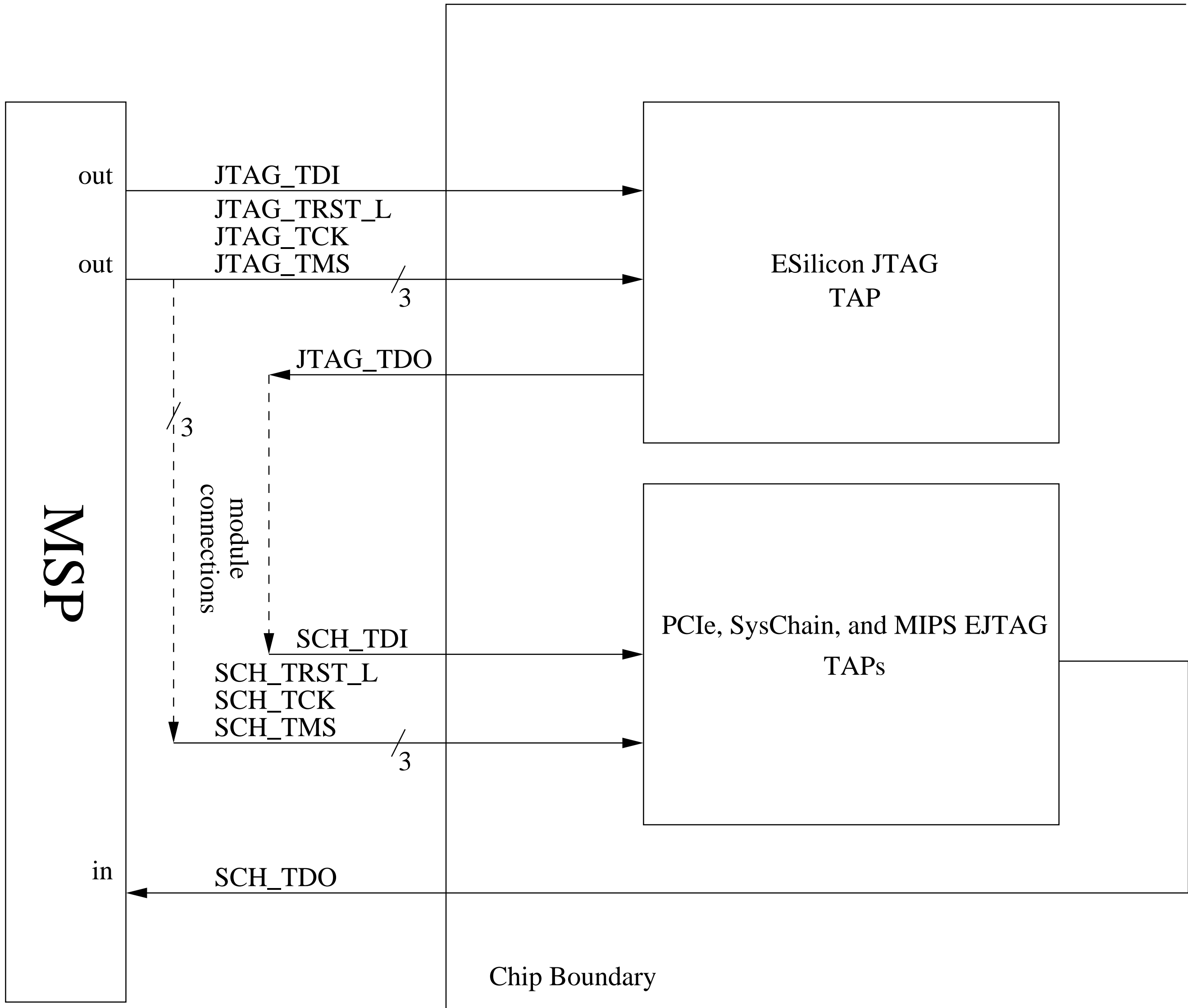


Figure 12.8: ICE9 E-Silicon and SysChain JTAG TAP Connections

SysChain's Reset Control Register (See Section 12.6.10.) which OR into the reset distribution for the relevant parts of the chip. Distribution of hard-reset assertion is asynchronous; distribution of the de-assertion is synchronous within a PLL clock domain. Hard-reset is distributed to all resetable logic on the chip. Assertion of the soft-resets is synchronous to the SysChain_TCLK scan clock, which is asynchronous w.r.t. the clocks for the logic being reset. De-assertion of the soft-resets is synchronous after passing through the dual-rank synchronizer for the appropriate clock domain, like the de-assertion of hard-reset. Perhaps the Figure 12.9 will make things clear. The RCREG_RESET_CCLK[*] and PLLCREG_RESET_PLLC pins are signals from the SysChain reset vector (section 12.6.10), hence they are in the SysChain_TCLK scan clock domain. The 2 flops form a dual-rank synchronizer to bring the signals into, in this case, the cclk domain. The gates downstream provide an asynchronous path around the flops for the asserting edge, so that only the deasserting edge is synchronous in the cclk domain. For the PLL_resets (RESET_PLL_C in the figure), both edges must be asynchronous, since the clock will not be running to clock the flops until the deassertion of reset propagates to the PLL.

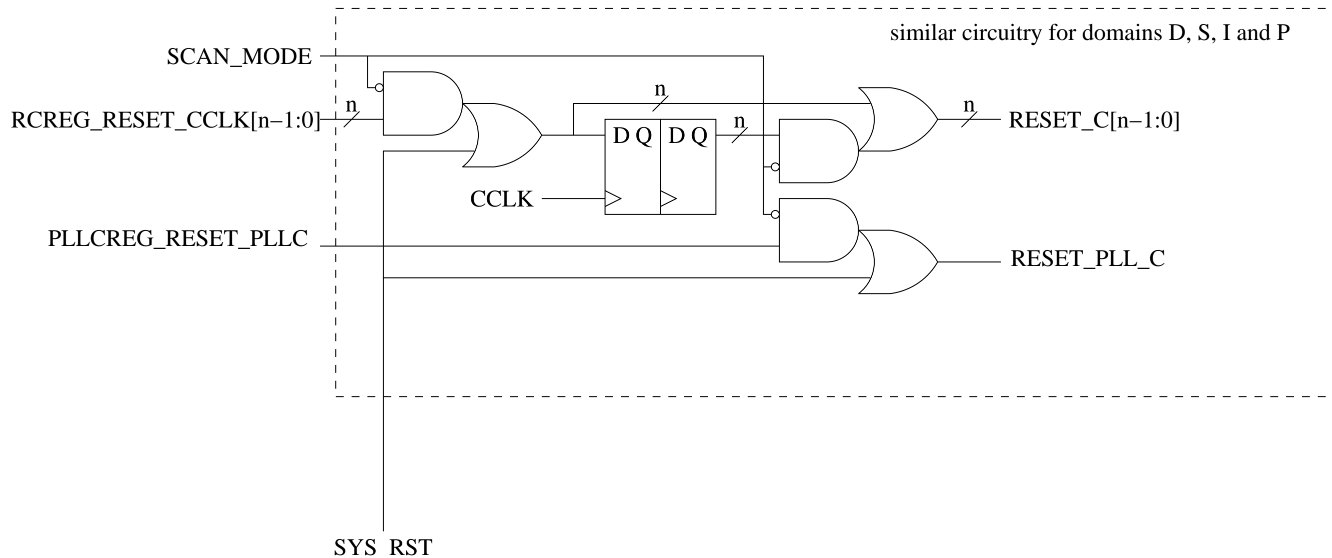


Figure 12.9: Reset Distribution for the CClk domain

For logical clarity the figure is drawn without any indication of signal assertion level. RESET_C[*] is the normal reset and would be used for most logic.

There will be a number of reset signals, one for each part of the chip which needs to be reset separately under control of the module service processor. The distribution of resets and clocks are shown in Figure 12.10.

12.8 Boot Timeline

This section describes the order of system bring-up from outlet-power.[4] Specifics on power sequencing, etc, may be found in the system specification.

12.8.1 SSP Boot Timeline

1. On power being applied to the cabinet, the first thing to power up and boot is the System Service Processor.
2. Whether automatically or on command from an administrator, the SSP enables power to the CPU modules.

12.8.2 MSP Boot Timeline

1. Once power is applied, the hard reset pin, sys_rst_l, and sch_trst_l are asserted to every ICE9. This is done with hardware even before the MSPs (module service processors) boot. The sys_clk is insured to be running

[4]Other documents reference the step numbers in the sections that follow. It is highly recommended that the ordering of existing steps remain unchanged. Adding steps to the end of a list is safe, but if additional steps must be inserted into the middle of a list, add them at an indented level as a,b,c,... etc. If a step must be removed from a list, keep the step, but replace its text with an italicized comment; such as: *This operation removed; continue to the next step.*

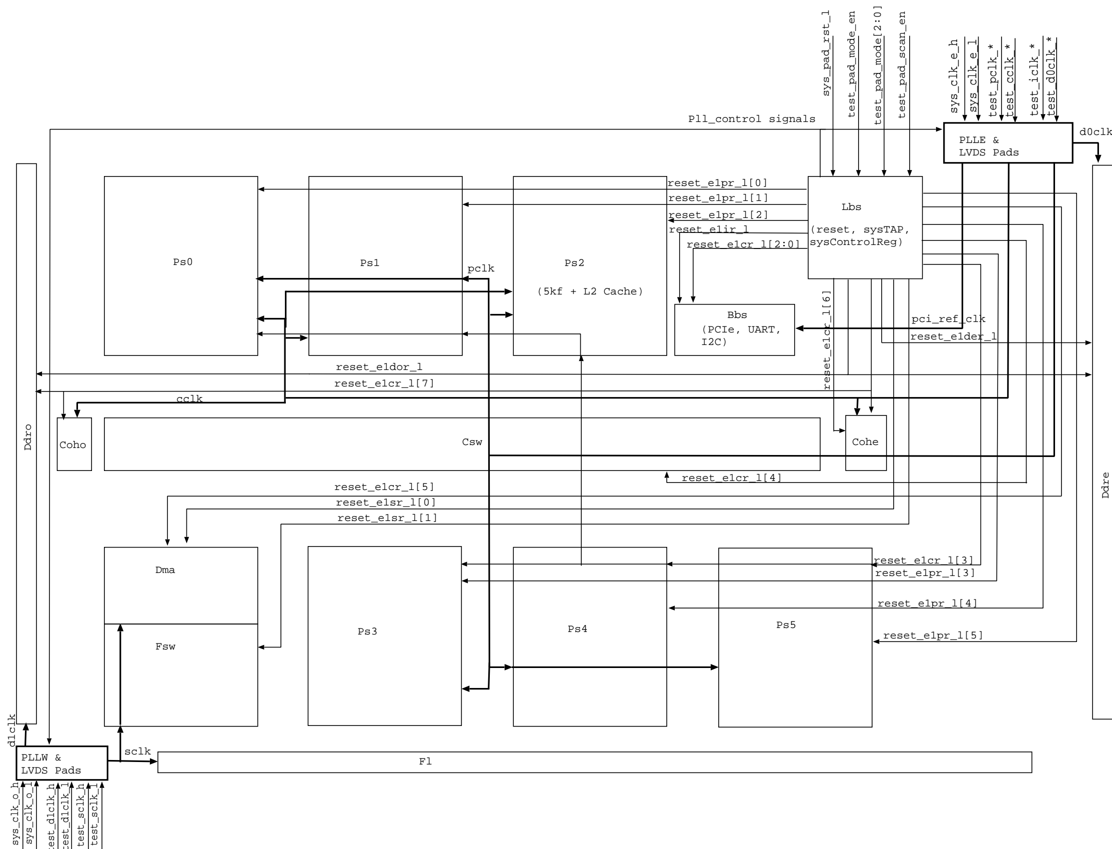


Figure 12.10: Reset & Clock distribution block diagram with real net names

so that sys_rst_l propagates throughout every ICE9 as described below.

2. Each MSP boots from its internal flash. The MSPs request a kernel and application from the SSP, which serves them via TFTP or similar mechanism.

3. Each MSP turns on the DC-DC converters that power the ICE9 chips on its module.

4. Each MSP begins an orderly bring-up of all the ICE9 chips on its module, in parallel.

12.8.3 Pre-DRAM Boot Timeline

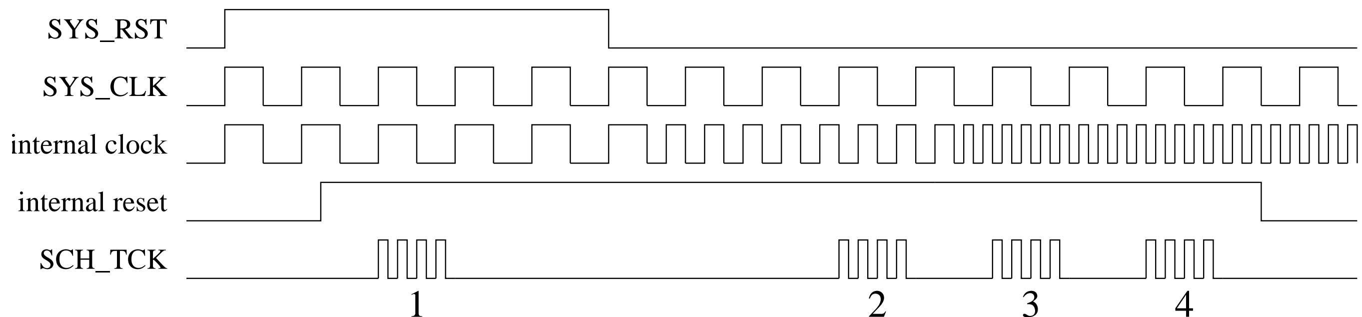


Figure 12.11: Reset Timing

1. The power-on assertion of sys_rst_l at the ICE9 has two effects on the ICE9. First, the PLL reference clocks, sys_clk_e_h/l and sys_clk_o_h/l, bypass their respective PLLs, so that all domains are clocked by sys_clk. Since most ICE9 resets are pipelined and are therefore effectively synchronous, this ensures that reset propagates throughout the chip. Second, sys_rst combinationally bypasses the sys_chain reset register, so that reset is applied without the need for any sys_chain scan activity. In figure 12.11, "internal_reset" represents a reset signal in any clock domain.

2. At some later time (sch_tck activity block 1 in figure 12.11), the MSP resets the sys_chain TAP controller and the six EJTAG TAP controllers, which share a common 4-wire TAP, by issuing two sequences of five TCK pulses with TMS asserted, followed by one TCK pulse with TMS de-asserted. This sequence guarantees that all TAP controllers in the sys_chain are reset and left in the RUNTEST/IDLE state and, by running the sequence twice, that the Virage Fuse-ROM values have been properly loaded. Every TCK pulse with sys_rst_l asserted initializes the PLL, Reset, CPU Debug Interrupt, and SMS RAM BIST control registers, and the SCB interface and Attention MSP registers, to their reset values.

3. Subsequently, the MSP stops asserting sys_rst_l. However, all internal resets remain asserted because of the initialization of the Reset control register. The PLLs are no longer bypassed with sys_clk; each will run at a frequency determined by the values with which the PLL Control register was initialized. The allows RAM BIST activity via ATE to proceed at an appropriate speed.

4. The MSP must poll the lock status of each PLL until either its lock bit is set or the MSP times out waiting for lock. This is shown as SCH_TCK activity block 2 in figure12.11. Prior to polling for lock, the MSP may at this time make changes to the PLL control values. The correct method for changing any PLL control value is to first assert reset to the PLL to be changed, changing the control value and then de-asserting reset to the PLL followed by polling its lock bit for assertion. Setting the PLL control registers can be skipped if the values established in step 3 are the desired values, however in all cases the MSP must poll each PLL for lock before proceeding to the next step.

5. Prior to Virage BIST, the MSP deasserts all SMS Resets in the Reset Control Register, leaving all normal resets asserted. The MSP then enables Virage BIST and waits for the results and reads them back (see section 12.6.13 for details). After successful BIST completion, the Virage RAMs will have been cleared and the MSP de-asserts the SMS CLK Enable bit in the Reset Control Register to prevent further BIST operation. This is shown as SCH_TCK activity block 3 in figure12.11.

(a) After Virage BIST, the MSP must bring the ICLK PLL out of reset. It is the only PLL that comes up held in reset state by assertion of sys_rst_l. To bring the ICLK PLL out of reset, the MSP must first insure that the PCI Reference Clock PLL is in lock (done in step 4 above). The MSP then must write 3'b1 to bits <57:55> (the Pllne field) of the sys_chain register R_SysTapPll to insure that the PCI Reference Clock is driven onto the ICE9 pins. The default value of this field after reset is 3'b0 which would leave the PCI Reference Clock pins in HighZ mode.

(b) After the MSP has set the R_SysTapPll register to deliver the PCI Reference Clock to the chip pins, it then de-asserts the IclkReset bit in the PLL control register and polls the IclkLock bit for assertion. At this point the ICLK PLL is operating normally and is no longer in reset.

6. The MSP sends an EJTAGBOOT instruction to each of the 6 processor EJTAG controllers. When reset is released in a later step, this will override the default fetch from 1FC0000 at reset, and instead immediately cause the CPU to take a debug exception and wait for instructions over EJTAG.

7. The MSP then deasserts the internal reset signals for all functional blocks (see 12.6.10). This is shown as SCH_TCK activity block 4 in Figure12.11.

   (a) Note: UartIoEnb is left de-asserted. This will be bundled into whatever code the MSP uses to mux Serial I/O to the ICE9s. Whenever a connection is opened to a particular ICE9, that chip's UartIoEnb will be asserted at that time. It will be de-asserted when the MSP closes the connection.

8. The MSP uses the SysChain/SCB interface to load the module number into R_ScbChipNum (see 10.14.7.)

9. The MSP scan in of EJTAGBOOT in step 6 and release of reset in step 7 causes the CPUs to wait for EJTAG instructions. The MSP sends the initial boot routine (boot0.s) to CPU 0 *only* and then force jumps it via EJTAG to the start of the boot0 image.

10. CPU 0 begins running the boot0 image. The boot0 routine initializes the register file, TLB and caches and copies the boot1 routine (boot1.s) from the MSP into the L2 cache. Boot0 then jumps to the boot1 image in the L2 cache.

11. Boot1 begins executing from the L2 cache. At this point the only memory-system difference from normal operation is that the DDI initializes in a mode which returns bogus data on reads; otherwise the normal L1/L2 write-allocate would hang on the first miss.

12. CPU 0 starts a memory copy loop, which reads from the EJTAG debug region and writes the L2 cache.

13. The MSP sends the second boot image to CPU 0 (boot2), using the FASTDATA EJTAG command. This requires ˜71 shifts per 64-bits of data, or ˜2.5 seconds for 256KB at 1 MHz sch_tck. The entire image is limited to the L2 cache size, or 256 KB; if this is exceeded the DRAM would need to be initialized before this loop to prevent the L2 from creating victims.

14. The MSP boot image also includes configuration data for the boot process, including PCI-connected and DRAM frequency information.

15. When the copy loop completes, CPU 0 executes the code. This image starts the next phase of the boot process.

12.8.4 DRAM Boot Timeline

1. The newly installed cache code initializes DRAM. This includes reading the DIMM I2C configuration, programming the controller, and testing/zeroing memory. (Of course, the code needs to be careful not to overwrite or evict itself until it competes the memory copy. One alternative is to have stage one boot load at 31GB - above where there will be memory.)

2. The code performs the BIOS-ish initalization required prior to kernel boot.

3. After DRAM is initialized, the boot0/boot1 step described for CPU 0 is repeated on CPUs 1-5. The download copy loop steps for boot1 are skipped, as the boot1 image is already in the L2 cache. CPUs 1-5 jump directly to boot1 at the end of boot0.

4. Now running from the caches, CPUs 1-5 enable interrupts and execute a WAIT instruction, which will put them to sleep until they receive an interrupt from CPU 0 during kernel boot.

5. The kernel loader is copied into DRAM by CPU 0 from the MSP via the EJTAG FASTDATA command. At the completion of the copy, the MSP force jumps the CPU 0 into the kernel loader. Unlike the previous memory copy loops, the kernel loader performs decompression and checksumming of the kernel.

6. The MSP receives the compressed kernel image fromthe SSP and uses the EJTAG FASTDATA command to transfer it to the kernel loader running on CPU 0.

7. Upon successful completion of the kernel download, it is executed.

12.8.5 Kernel Boot Timeline

1. The kernel performs its normal boot sequence.

2. When kernel boot is complete, CPU 0 sends a interrupt to CPUs 1-5, which releases them from WAIT.

3. The kernel asks the MSP to enable the watchdog timer. (Or, more correctly, switch from a very long wait-for-boot timeout to a shorter heartbeat timeout.)

4. The fabric and DMA drivers initialize the fabric switch, links, and DMA engine as described below.

5. Login :)

12.8.6 Booting the Fabric Switch and Links

1. At power-on, the fabric links, fabric switch, and DMA are held in reset by bits in the R_SysTapReset register. Deassert reset to FSW (clear FabSw bit in R_SysTapReset).

2. Configure the FSW registers through writes on the SCB. See the FSW chapter for details on each register. In particular, in R_FswBlockReset, deassert reset on all blocks. In R_FswBlockEnable, enable all blocks. The FSW is now ready to transfer packets to/from the links and DMA, but nothing will happen yet since the links and DMA are still in reset.

3. Bring fabric links out of reset (clear FabLn bit in R_SysTapReset).

4. Configure the FL registers through writes on the SCB. Bring up each link into MissionMode. See the Fabric Link chapter for details.

At this point, the ICE9 can accept packets from its three upstream neighbors and send them to its three downstream neighbors. The MSP or a processor can use out-of-band communication channels, watch packet statistics, set and clear interrupts, etc. This ICE9's DMA engine cannot send any packets because it is still in reset. Any packets coming from upstream that are destined for the DMA flow through the fabric switch to the DMA RX port, which because it is in reset, will accept the packets and drop them.

On nodes with BIST, DRAM and other failures preventing Linux boot, the MSP will be able to initalize the fabric by this process using the SysChain/SCB alone (without any cpu core).

12.8.7 Booting the DMA Engine

It is assumed that the fabric switch and links are already initialized as described in the previous section.

For the processors to communicate with the fabric (other than reading and writing CSRs), they must boot the DMA engine. The DMA engine must be configured by processors, because many of the configuration registers are accessible only through the CSW.

1. Bring the DMA engine out of reset (clear Dma bit in R_SysTapReset).

2. Configure the DMA engine

   (a) Write zero to every location in R_DmaDmem and R_DmaImem.

   (b) Write DMA microcode to R_DmaImem and initial data to R_DmaDmem from the DMA loader file.

(c) Initialize the DMA microcode application data as described in the Initialization section of the DMA chapter. For example, the application needs the physical address of various queues and data structures.

3. Start the DMA Engine by setting all ThreadEnable bits in R_DmaThreadSel.

4. Deassert reset to the DMA's TX and RX ports in R_DmaBlockReset. This allows packets to begin to flow between the DMA and fabric switch.

12.8.8 Rebooting with Fabric Still Up

The ICE9 allows the fabric switch and links to be operated even while the rest of the node is being reset. As long as the FabSw and FswLn bits of R_SysTapReset are deasserted, the fabric switch and link will continue to route fabric traffic. This allows the ICE9 to be rebooted without backing up the fabric. When software has decided to reset the chip without affecting the fabric, the sequence of events is as follows:

1. Disable the crosspoint buffers in the fabric switch leading from DMA to the fabric transmitters by clearing R_FswBlockEnable bits for XB30, XB31, and XB32. This prevents any new DMA traffic from flowing into the fabric. (Using R_FswBlockEnable instead of R_FswBlockReset stops traffic on clean packet boundaries.)

2. First shut down the DMA (cleanly if possible) and then assert reset to the DMA. Once the DMA is in reset, its RX port accepts incoming packets and throws them away, and its TX port will not send anything else.

3. Reset anything else in the chip that is needed. At the point in the boot process that the fabric switch would be initialized, you need to detect whether the fabric switch and link are already running. For example, the detection could be based on whether FSW and FL are already out of reset, or if fabric links are in mission mode, or it could be based on nonzero packet counter statistics. If the FSW and FL are not running, you would initialize them as described in section 12.8.6. If they are already running, continue with this sequence.

4. Enable all blocks in R_FswBlockEnable.

5. Proceed with Booting the DMA Engine, described in section 12.8.7.

Chapter 13

PCI Express Subsystem

[$Id: chippci.lyx 50693 2008-02-07 16:01:46Z wsnyder $]

13.1 Overview

The ICE9 chip includes a PCI-Express root complex subsystem. The PCI-Express subsystem provides the ICE9 cores with access to PCI-Express peripheral chips either on the processing module or on external cards. While the subsystem typically talks to just a single PCI-Express device, there is no hardware limitation that prevents implementation of more complex topologies.

The specifications for tphycr
he PCI-Express subsystem are:

- Implements a root complex.
- Supports packet sizes of 128B, 256B, and 512B.
- Supports end-to-end CRC checking.
- Supports one virtual circuit.
- Supports PCI-Express power off mode L3.
- Translates CPU physical addresses to/from PCI addresses. See Chapter 16.

13.2 Differences, Bugs, and Enhancements

13.2.1 Product and Chip Pass Differences

1. ICE9B fixes legacy interrupt D behavior incorrect during a link down, bug1984. In ICE9A if an ASSERT_INTD message arrives from the endpoint, software will service the interrupt. During this time, if the link goes down, an implicit DEASSERT_INTD should occur, but this did not happen. So if the interrupt service routine ends with a "wait for DEASSERT_INTD", and it is possible that it will hang forever.

2. ICE9B fixes ecc error ignored when CLEAR comes at the same time, bug2028. In ICE9A if an ECC error is in effect and the interrupt is raised. Some time software clears the interrupt and an ECC error comes at the same time (in PMI where is checks, or not checks, for ecc error and clear), PMI ignores the second ECC error.

3. ICE9B fixes the MsiBaseAddr register addressing, bug2097. In ICE9A, software has to program the PMI MsiBaseAddr register with an Ice9 address converted into a PCIe space address (look at the address mapping in the hardware spec).

4. ICE9B fixes RX detection not being completed when some lanes are disabled, bug2113. In ICE9A, when one or more lanes of a multi-lane link are disabled using TxCompliance/TxElecIdle as described in Section 8 of the PIPE specification, initiating a receiver detection sequence will cause the PCS layer to hang due to the

”turned off” lanes not performing the receiver detection operation. To workaround, enable all lanes prior to performing a receiver detection operation, as lanes which are turned off will not participate in the receiver detection sequence.

5. NEED IMPL: TWC9A fixes only the bottom 16 bit being writable in R_PmiVmReqDat, bug2760. We couldn’t find any PCIe vendor which uses vendor messages, so this is of only minor concern.

13.2.2 Known Bugs and Possible Enhancements

1. None.

13.3 Internal Structure

The PCI-Express subsystem consists of six layers:

1. The PHY layer, which implements the 2.5Ghz SerDes used for PCI-Express I/O.
2. The PCS layer, which converts parallel data binary data received from the MAC layer to 8B/10B encoded serial data for the PHY.
3. The MAC layer, which implements the physical connection path for PCI-Express.
4. The link layer, which implements the logical connection path for PCI-Express.
5. The transaction layer, which implements PCI-Express transactions and queues
6. An application layer, which interfaces between the L2 cache and the transaction layer.

Layer 6, the application layer, is designed by SiCortex, and is synthesized RTL. Layers 3-5, the transaction, link, and MAC layers, are part of the PCI-Express controller core. This core is purchased from Synopsys and is synthesized RTL. Layers 1-2, the PCS and PHY layers, are part of the PCI-Express PHY core. This core is purchased from Synopsys. The PCS layer is synthesized RTL. The PHY layer is a hard macro.

13.4 Known Bugs and Enhancements

1. R_SysTapReset_Scb was originally intended to reset only the SCB and the OCLA LAC. However, in ICE9A, ICE9A1 and ICE9B this also ends up resetting the cclk parts of the PMI. This was not intended. In future revisions of the chip an additional bit may be added to the R_SysTapReset register to allow for resetting the PMI without resetting the SCB and OCLA LAC. bug2929.

13.5 Process Requirements

The PCI-Express PHY core requires 2.5V or 3.3V thick oxide and input voltage for its analog circuits. For its 90nm general purpose (G) process, TSMC offers either a dual oxide option (1.0V/2.5V) or a triple oxide option (1.0V/1.8V/3.3V). Since the DDR PHY is a dual-process DDR/DDR2, 2.5V/1.8V design, it does not require 1.8V oxide, and ICE9 will use the dual oxide option; thus the PCI-Express PHY will run off 2.5V, as will all general-purpose IO buffers and PLLs.

13.6 Application Layer and the PMI

PMI is the unit name for the PCI controller and all application layer components. This unit also includes the interface between the L2 cache switch (CSW) and the miscellaneous I/O units including the UART, I2C and the SCB. Figure 13.1 shows the top level block diagram of the application layer and its connection to the root complex.

The PMI is comprised of 5 major pieces. The CSI is a control/status register interface that allows processors to perform I/O register reads and writes to the UART (see Section 15), the I2C controller (Section 14), the Serial Control Bus (Section 10), the RC’s configuration port (DBI), the RC’s vendor message interface (VMI), the RC’s system information interface (SII), the PCI PHY configuration port and internal control and status registers. The

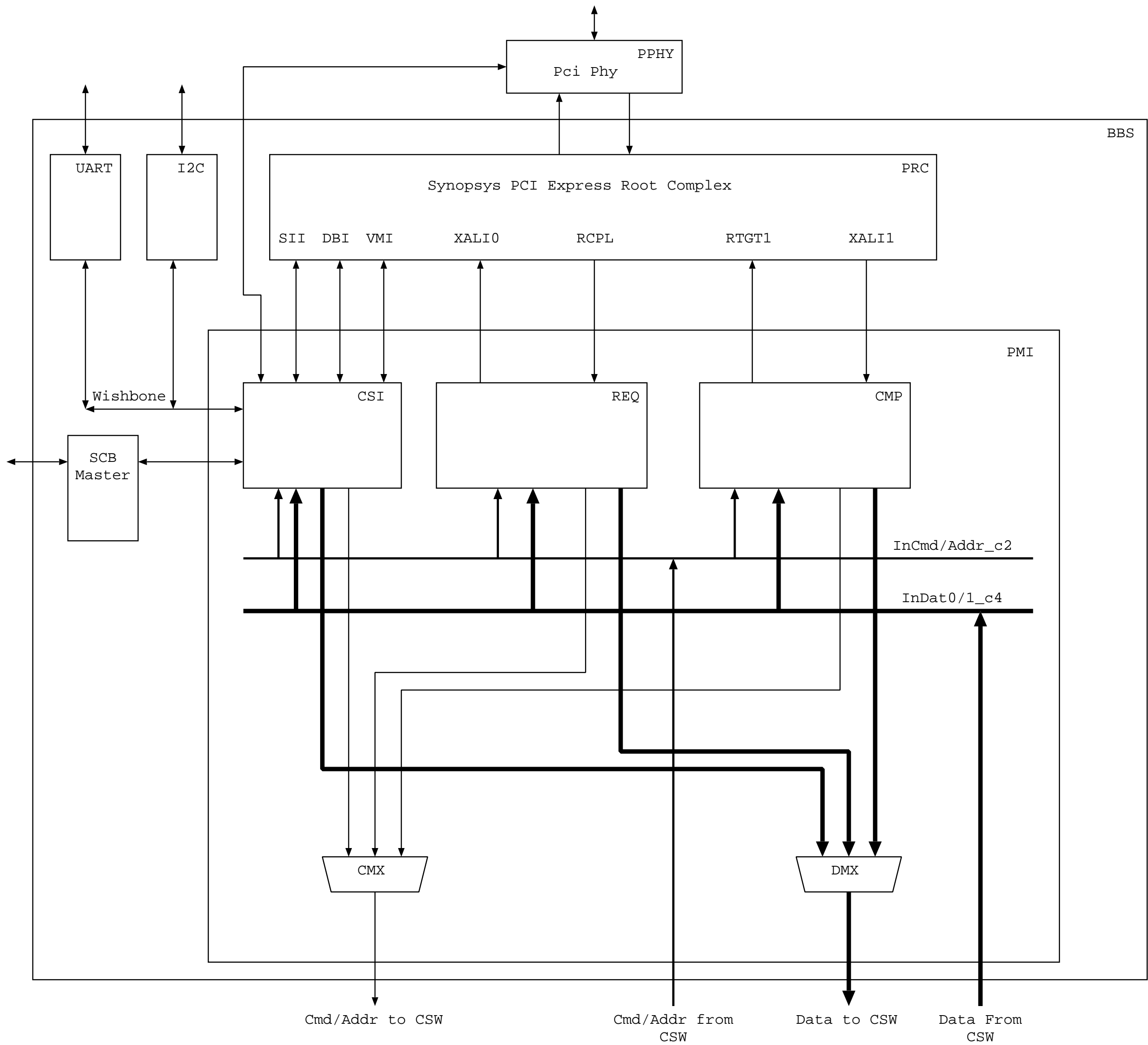


Figure 13.1: PMI Block Diagram – The Application layer between the CSW and PCI Root Complex

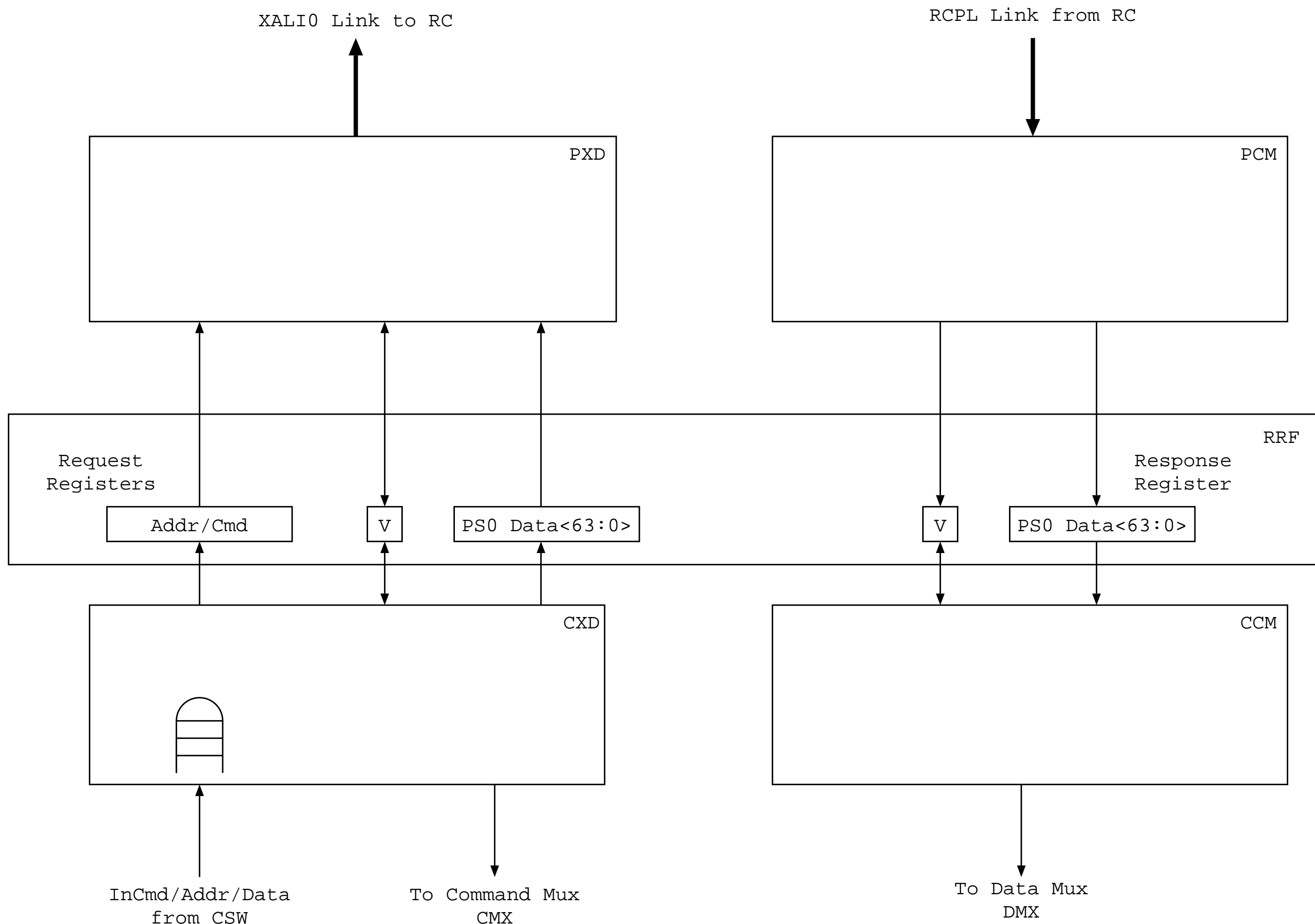


Figure 13.2: REQ Unit

REQ handles all requests from processors and the responses generated by the PCI network. The CMP handles inbound requests from downstream devices and generates completion events in response to the requests. The CMX is the command multiplexer and the DMX the data multiplexer. Each of these components are described below.

It is important to note that the PMI contains logic that runs in TWO different clock domains. The RC is driven by a fixed frequency ICLK at 250MHz. The PMI interface to the CSW runs at that CCLK frequency that may range from 200 to 300 MHz, as it is tied to the processor clock rate. The synchronizer boundaries between the two domains are contained entirely in the CSI, REQ, and CMP units.

13.6.1 The Requestor Unit REQ

The requester unit transforms CSW I/O accesses (RDIO and WTIO) into PCI Express Transaction Layer Packets (TLPs). In the case of read transactions, the REQ also handles the returning completion TLP from the RC and turns it into a 64 bit data transfer over the CSW back to the original requesting processor. RDIO and WTIO requests are limited to no more than 64 bits. As such, only CSW Data0 and the associated byte mask are relevant. The REQ generates six kinds of TLPs: Memory Read, Memory Write, IO Read, IO Write, Config Read and Config Write. In the case of all non-posted requests, the PCI TID assigned to the transaction is equal to the TID received on the CSW command/address TID inputs. This allows simple matching of completion events to the initiating request.

A block diagram of the REQ is shown in Figure 13.2.

13.6.1.1 REQ Memory Read Request Handling

A memory read request is initiated by a CSW RDIO to an address in the PCI Memory Address Range. The RDIO CSW operation arrives on the inbound command bus. It is then converted into a transaction on the XALI0

interface that will create a memory read request transaction. At some later time, the RC will respond with a completion packet on the RCPL port. This will be converted by the REQ into a transaction on the CSW Data lines. All memory read requests from the CSW are 64 bit aligned. Addresses on the PCI, however, can be 32 bit aligned. As such, if the active bits in the byte mask indicate that the TLP can be contained within a 32 bit aligned chunk of data, the address will be modified to be 32 bit aligned and only 4 bytes will be retrieved across the PCI. Returned data will either occupy all 64 bits of the Data0 lines on the CSW, in the case of a 64 bit access, or the 32 bits retrieved will be duplicated on the upper and lower 32 bits of Data0. Requests to addresses within the first 4GB of the PCI Memory Address range will cause 3 DW header (32 bit address) transactions, while those to the remainder of the range will cause 4 DW headers (64 bit address) transactions.

13.6.1.2 REQ Memory Write Request Handling

A memory write request is initiated by a CSW WTIO to an address in the PCI Memory Address Range. The WTIO CSW operation arrives on the inbound command bus. The REQ responds by initiating a RDIO CSW operation to retrieve the write data from the original requesting processor. Once the data has arrived, the REQ builds a memory write TLP by wiggling the appropriate signals on the XALI0 interface to create a memory write transaction with the appropriate byte mask. Like read requests, write requests are aligned to 64 bits. However, if the data to be written is contained within one 32 bit aligned chunk, as indicated by the byte mask, the address will be modified to be 32 bit aligned and only 4 bytes of data will be sent. Requests to addresses within the first 4GB of the PCI Memory Address range will cause 3 DW header (32 bit address) transactions, while those to the remainder of the range will cause 4 DW headers (64 bit address) transactions.

13.6.1.3 REQ IO Read Request Handling

An IO read request is initiated by a CSW RDIO to an address in the PCI I/O Address Range. Other than the transaction type field driven to the RC, the REQ processes an IO Read Request in the same manner as a memory read request. IO requests are, however, limited to no more than 32 bits of data and address (this means the byte mask for Data0 from the CSW can only have bits set in the upper or lower nibble). The address is appropriately modified as per the bits in the bit mask.

13.6.1.4 REQ IO Write Request Handling

An IO write request is initiated by a CSW WTIO to an address in the PCI I/O Address Range. Other than the transaction type field driven to the RC, the REQ processes an IO Write Request in the same manner as a memory write request. IO requests are, however, limited to no more than 32 bits of data and address (this means the byte mask for Data0 from the CSW can only have bits set in the upper or lower nibble). The address is appropriately modified as per the bits in the bit mask.

13.6.1.5 REQ Configuration Read Request Handling

A config read request is initiated by a CSW RDIO to an address in the PCI Configuration Address Range. The REQ processes a Configuration Read Request in a similar manner as a memory read request. The transaction type is different and the address is modified to shift bits [27:12] up to bits [31:16]. The address is also appropriately modified to account for config transactions being 32 bit aligned. If bits [27:20] in the CSW address match the primary bus number of the RC, an error will be returned to the originator. This signifies an attempt to access the RC config registers. Accesses of the config register within the RC can only be made via the DBI. If bits [27:20] in the CSW address match the secondary bus number, a CONFIG0 type transaction will be sent. Any other values will be sent as a CONFIG1 type transaction.

13.6.1.6 REQ Configuration Write Request Handling

A config write request is initiated by a CSW WTIO to an address in the PCI Configuration Address Range. The REQ processes a Configuration Write Request in the similar manner as a memory write request. The transaction type is different and the address is modified to shift bits [27:12] up to bits [31:16]. The address is also appropriately modified to account for config transactions being 32 bit aligned. If bits [27:20] in the CSW address match the primary bus number of the RC, an error will be returned to the originator. This signifies an attempt to access the RC config registers. Accesses of the config register within the RC can only be made via the DBI. If bits [27:20] in

the CSW address match the secondary bus number, a CONFIG0 type transaction will be sent. Any other values will be sent as a CONFIG1 type transaction.

13.6.1.7 REQ Sub-blocks

The REQ spans both the CCLK and the ICLK domains. The CXD and CCM both operate in the CCLK domain. The PXD and PCM operate in the ICLK domain. The RRF handles all clock domain crossings.

Commands from the CSW are pushed into a FIFO within the CXD. The FIFO is six entries deep (one for each of the six processors – we don't allow the DMA engine to send transactions to the RC). Commands are taken off the FIFO one at a time and fully processed before the next command is attended to. The CXD is responsible for decoding the incoming address (to determine which address region – PCI Memory, PCI I/O, or PCI Configuration – the address maps into) and sending the command/data to the RRF. If the operation is a write operation, the CXD must issue a RDIO command to first fetch the data payload and will write the command and data once the RDIO data is received. Read commands are sent to the RRF directly after address decoding.

The PCI Express side of the transmit path (the PXD) reads the command/data from the RRF. The PXD converts the Address, Command, Byte mask, and TID from the CSW into the appropriate outbound packet via the XTALI0 bus to the RC.

Completion packets arrive on the RCPL port from the root complex and go to the PCM. The PCM rips the reply packet apart and writes the returned 64/32 bit word and transaction ID into the RRF. A completion can not be serviced until all write transactions that preceded it coming from the RC have been completed. The CCM takes the data from the RRF and passes it to the DMX, in the case of read operations. It also sends a release to the CXD for all completions, allowing it to move onto the next command.

13.6.1.8 REQ Exception Handling

Errors conditions can arise in a number of places in the REQ:

Errored Completion from Root Complex If the RC signals an error in a completion, the error details will be logged in the PmiReqCompErr register (section 13.13.15) and a bit set in the PmiIntr register (section 13.13.2). Sources of this error include bad ecrc, poisoned, unsupported request, completer abort, config retry, tlp abort, dllp abort and completion timeout. The PmiReqCompErr register includes information containing the reason for the failed completion.

If the transaction was a read, all ones data will be returned to the originating processor. The exception to this is a Config Read with an “unsupported request” completion; this is a normal part of the enumeration process and so all ones data will be returned, but no error logged.

It is expected that in the event of a config retry, the originating processor will reissue the config command after a suitable delay as required by the PCI Express specification.

Data with Bad ECC from CSW If data with an ECC error arrives from the CSW, the error details will be logged in the PmiReqEccErr register (section 13.13.14) and a bit set in the PmiIntr register (section 13.13.2). The transaction will be completed regardless of whether the error was of a single bit or double bit nature.

13.6.1.9 RC Config Register Access

The Requester unit does NOT support the legacy I/O based configuration mechanism present in some earlier personal computer based implementations of PCI root complexes. That is, we don't support the “PCI Compatible Configuration Mechanism” using I/O addresses 0CF8 and 0CFC. All configuration transactions to non-RC devices are via the PCI Express Enhanced Configuration Mechanism. The RC config registers can only be accesed via the DBI interface. See Section 13.6.3.

13.6.2 The Completer Unit CMP

The Completer Unit is responsible for handling incoming requests from downstream PCI Express devices. The primary goal in the design of the CMP is to maximize the available bandwidth. We are not necessarily aiming for low latency; we'll trade latency for more bandwidth whenever we get the chance. The PMI must support an aggregate bandwidth of 2GB/s in each direction to keep the link fully busy.

The CMP handles three transaction types: Memory Write, Memory Read and Message Signalled Interrupt operations. Each is first handled in the ICLK domain where the incoming completion or request packet is disassembled and digested. The digested form is then sent to a component in the CCLK domain where it is converted into a command or sequence of commands on the CSW. Data and header information for read requests are sent back into the ICLK domain to be sent along to the RC.

13.6.2.1 Memory Write Operation

When a downstream device on the PCI Express bus writes a block of memory, the data item may range in size from a single byte up to a 512B block. (We are capping the size to 512B within the RC). The data may or may not be aligned to a 64 byte boundary. Figure 13.3 shows the major blocks that participate in serving memory write operations. Note that PCI Express MemWrites are *posted* operations, so that no response is required on the part of the application layer.

Memory write operations are first fielded by the SYC, which is shared between the memory write and memory read logic. The payload is written into a data FIFO. The data is aligned to 128 bit boundaries, as found on the CSW, before it is written. The SYC also writes the byte masks, the start address, and data block length into the write command FIFO. When either the data or command FIFOs are full the SYC will assert a flow control halt signal back to the RC to stall the incoming request bus. All of this is done in the ICLK domain.

The CCW pulls the header/data from the FIFOs. The domain crossing from the ICLK to the CCLK is handled by this action. In the case of data blocks that are correctly aligned, the CCW will initiate a BWT operation for each 64 byte block in the incoming payload. It is important that we keep the data writes in order. For this reason and in order to prevent deadlock conditions, the CCW will not send out the command for a BWT to block X+1 until it has seen the BWTGO response for the BWT operation on block X. This may limit a single PCI device to less than the 4GB available bandwidth on the CSW data bus.

For blocks that are not naturally aligned or are less than 64B, the CCW must perform a write merge. The CCW will launch a RDEX operation for the initial block of data, a WINV to return the merged data, BWT operations as required for intermediate data and a final RDEX/WINV as required at the end. Each of these steps are handled serially and therefore only one write 64B merge buffer is required. The performance of transactions requiring merges will be much less than aligned transfers. Write requests from the PCI must be allowed to pass read requests from the PCI to forestall deadlock conditions.

The posted data buffer in the RC will be ECC protected. In the event of an uncorrectable error, status registers will record the syndrome and address associated with the error and a slow interrupt will be generated if enabled. The write will otherwise proceed as if un-errorred. If RDEX merge data has an uncorrectable error, the address and syndrome associated with the error will be recorded and a slow interrupt will be generated if enabled. Control registers allow the purposeful corruption of the data coming from the RC posted data buffer and written to the write data FIFO in the SYC.

13.6.2.2 Memory Read Operation

Memory read operations are non-posted transactions, so a completion is required. The memory read logic is shown in Figure 13.4.

Incoming read requests arrive at the SYC via the RTRGT1 port – the same port that carries write requests and MSI delivery packets. The SYC receives the incoming read requests and places them into a read request FIFO. This is done in the ICLK domain.

The request is pulled from the FIFO by the CCR, thereby effecting the clock domain crossing to the CCLK. A request can not be serviced until all write requests that preceeded it out of the RC have been completed by the CCW. The request is parsed into one or more BRD operations. If the request begins or ends at an unaligned address, the unneeded data from the first and last BRDs will be discarded prior to being presented to the SYC and written into the Completion Data FIFO. This weeding is done on 128 bit quanta.

Up to three BRDs can be in flight at any one time. The data associated with the BRDs need not come back from the CSW in the order they were requested, but they must be presented to the SYC in order. Three buffers within the CCR first accept the data from the CSW as it arrives. A separate state machine reads the data from these buffers and sends it to the SYC in the needed order. The "weeding" mentioned above is done at this point.

At the time the request service begins, the request information is also written into the Completion header FIFO in the SYC. A state machine in the SYC services each request in turn, generating the appropriate PCI transactions. Servicing of a completion header begins by determining if a split completion is required, what the data alignment is and how much data is required in quanta of 128 bits. While the PCI Express requests may ask for up to 4KBytes

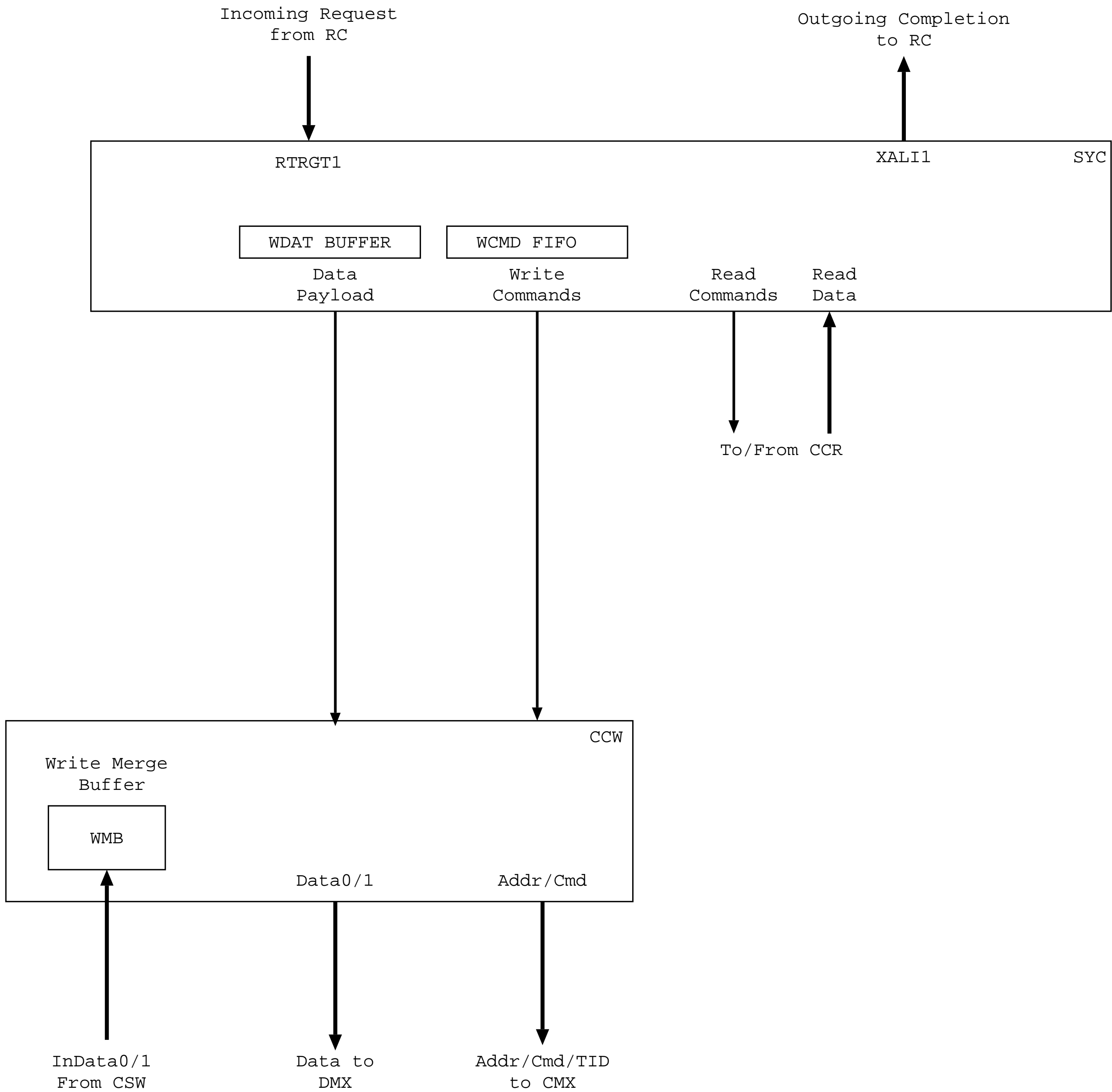


Figure 13.3: Memory Write Machinery

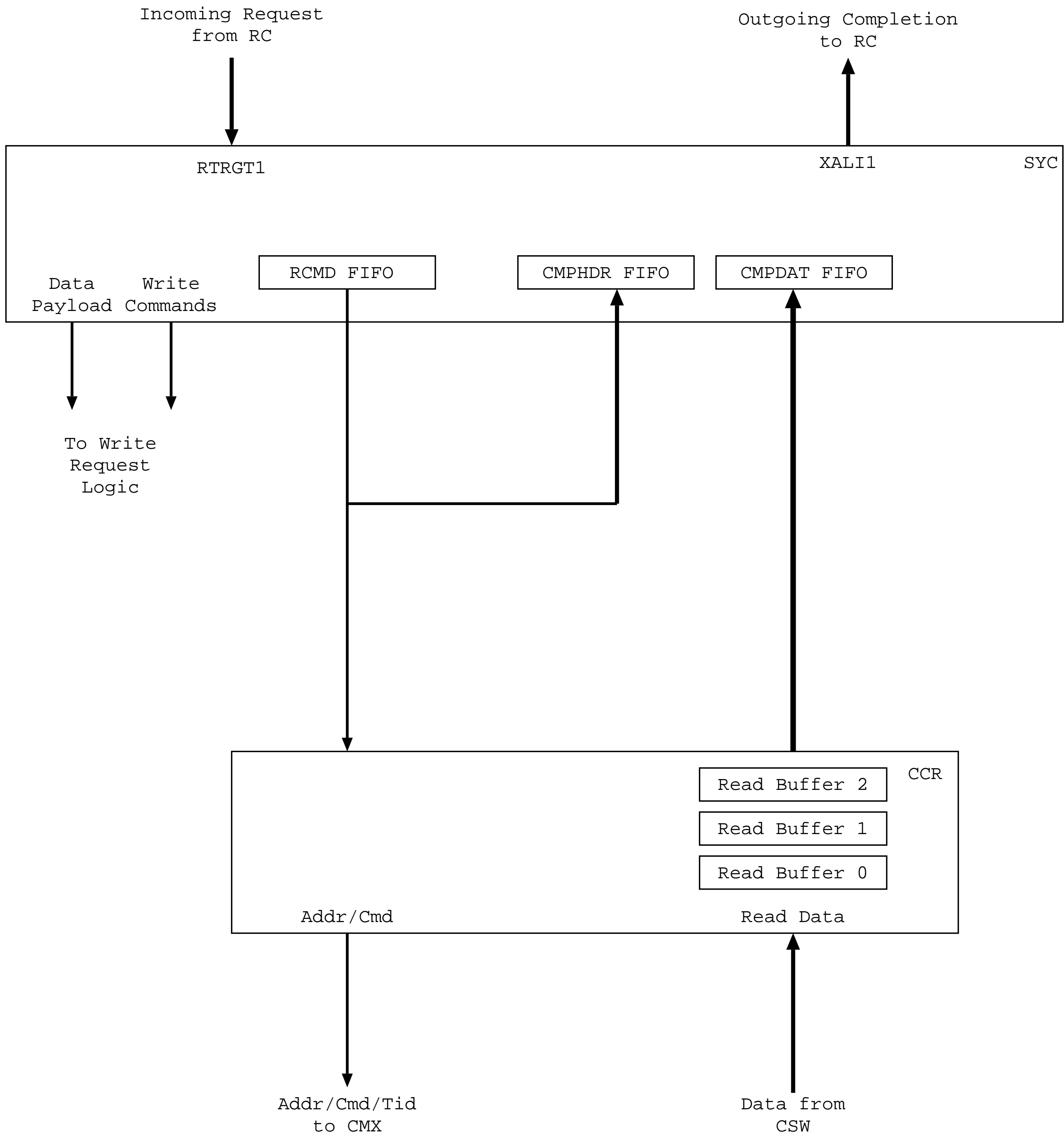


Figure 13.4: Memory Read Machinery

in a single transaction, we limit our completions to 512B. The actual size is set by the Max_Payload_Size register in the RC. When the correct amount of data is present in the Completion data FIFO, a completion is sent to the RC via the XALI1 interface.

Data coming from the CSW is ECC protected. The data with ECC is forwarded to the Completion Data FIFO without being checked. When read out of this FIFO, the ECC is checked and a "bad EOT" signalled to the RC in the event of an uncorrectable error. Status registers will record the syndrome and address associated with the error and a slow interrupt will be generated if enabled.

13.6.2.3 Message Signalled Interrupts

MSI interrupts are implemented by PCI Express devices as memory write transactions to an address that was initially written by the configuration software. That is, each device capable of initiating an MSI interrupt has a message address register to which it will write to signal the interrupt. Each such device also has a 16 bit message data register that will be written to the message address when the interrupt is signalled.

That fits rather nicely in with the interrupt scheme implemented in the ICE9 processor segment. Interrupts are delivered to a processor via the CSW INTR transaction that writes a 16 bit value to an interrupt cause FIFO. The low three bits (the intsel or interrupt select field) of the interrupt designate which of the six interrupts is to be signalled. The upper 13 bits (the reason field) contain any information the device requires to identify the reason for the interrupt.

So, the MSI scheme is rather simple. When the CCW detects a memory write to an address range specified by the PmiMsiAddr register (section 13.13.19), it generates a CSW INTR command to the processor (address bus stop) identified by address bits 5:2. The "address" associated with this command are the low 12 bits from the write data payload.

The MSI INTR command always uses TID PCIWT3.

13.6.3 The Control/Status Widget CSI

The control/status widget implements the interface between the CSW and the DBI/SII/VMI ports on the root complex, as well as supporting access to the Serial Configuration Bus controller, the PCI Express Phy, internal PMI configuration registers, and the 16550 UART. The CSI is shown in Figure 13.5.

Commands from the CSW are pushed into a FIFO. The FIFO is six entries deep, one for each of the six processors. Commands are taken off the FIFO one at a time and fully processed before the next command is attended to. The CSI processes only RDIO and WTIO commands from the CSW command/address bus. In the case of a RDIO, it will read the appropriate data register from the target and return the data. In the case of a WTIO, the CSI will initiate a RDIO command to the processor that issued the WTIO so as to acquire the write data. When the RDIO completes, the write data will be written into the target register.

The CSI is comprised of the WBI, DBI, CIF, CRI and CIN sub-blocks. The WBI is the wishbone bus interface to the UART and I2C. The DBI accesses the interface of the same name on the RC. The CIF contains the CSW command FIFO and handles the interfacing to the CSW. The CRI handles the interface to the Phy. The CIN contains the PMI internal status and control registers as well as allowing access to the RC SII and VMI signals. It also handles the slow interrupt generation.

13.6.3.1 The CSW Interface CIF

The CIF executes the CSW protocol. It accepts commands from the CSW and places them into a FIFO. The commands are pulled from the FIFO and parsed to determine if a RDIO back to the originating processor is required and also to determine which sub-function within the CSI should receive the command/data. The appropriate sub-function is sent the request and an ack awaited before moving directly onto the next command, in the event of a write, or sending the data back to the CSW and awaiting a CSW grant, in the event of a read, before moving onto the next command.

13.6.3.2 The Wishbone Interface WBI

The WBI receives requests from the CIF and translates them into the wishbone protocol. It awaits an ack from a wishbone device (the UART or I2C) and signals the CIF that the request has been completed. In the event that an ack is not received in a timely fashion, a completion is sent back to the CIF anyway. If the request was a read, all ones data is returned with the completion. The number of clock ticks until a timeout occurs is under software control via the PmiWbToVal register (section 13.13.20).

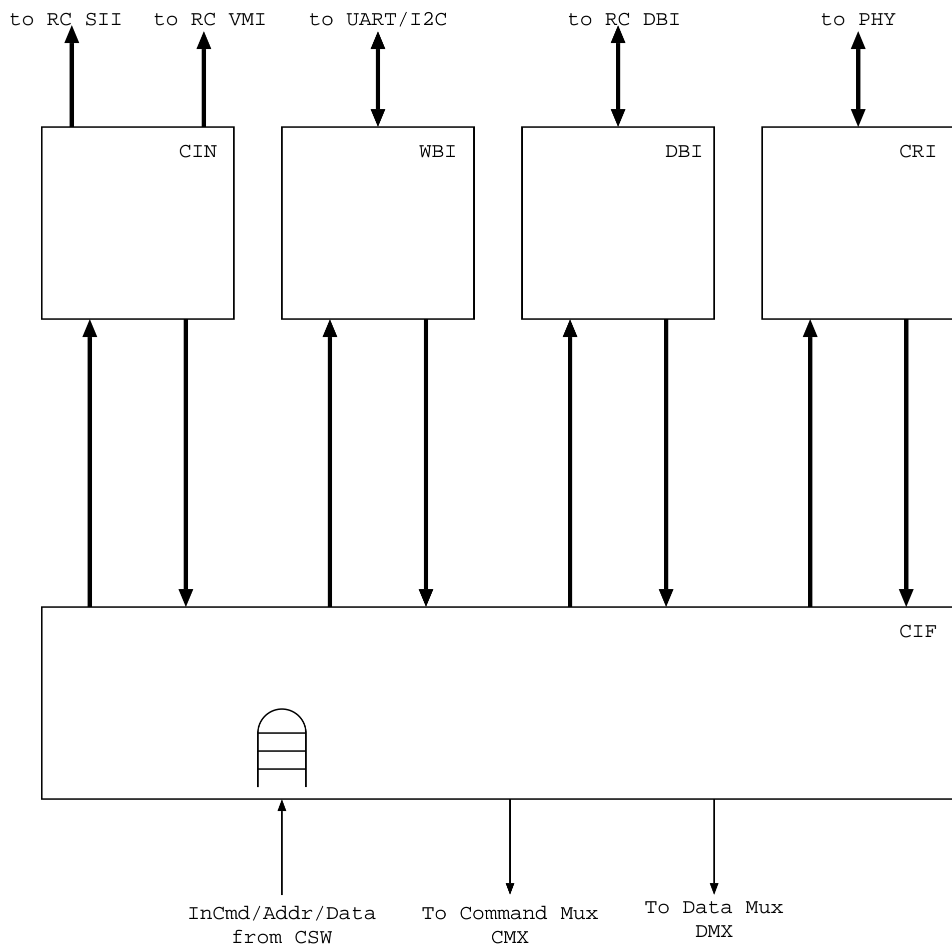


Figure 13.5: The Control/Status Widget

13.6.3.3 The RC Register Interface DBI

The DBI receives requests from the CIF and translates them into the data bus interface protocol as defined in the Synopsys Root Complex documentation. All configuration header space and extended configuration header space registers that pertain to the RC, are only accessable via this interface. The data bus interface includes a request/ack handshake. When the ack occurs, a completion is signalled back to the CIF, with or without data. This interface requires a clock crossing from the CCLK to the ICLK domains for a request and from the ICLK to the CCLK domains for the ack/read data.

13.6.3.4 The Phy Interface CRI

The CRI receives requests from the CIF and translates them into the interface protocol as defined in the Phy Core documentation. The interface includes a request/ack handshake. When the final ack occurs, a completion is signalled back to the CIF, with or without data. This is an asynchrounous interface; only the ack signal coming back from the PHY needs to syncronized to the CCLK.

13.6.3.5 The PMI Register Block CIN

The Cin performes a number of functions:

The CIN enacts the Vendor Message Interface (VMI) handshake. This is used to cause the RC to send a downstream vendor message. It is initiated by writing the appropriate data to the PmiVmReqDat (section 13.13.21), PmiReqHdr (section 13.13.22) and PmiVmReqCmd (section 13.13.23) registers. When the ack returns from RC, a completion is signalled to the CIF so that it can move onto the next command.

The CIF also aggregates all the SII (System information Interface) signals into a a number of registers. They are enumerated and described in the PMI Control and Status Register section (section 13.13).

All of the various error and status conditions that could cause a slow interrupt are aggregated into the PmiIntr register (section 13.13.2) withinn the CIN. The interrupt enable register PmiIntrEn (section 13.13.3) determines which of the potential sources can cause a slow interrupt. Some of the sources can be cleared directly by writing a one to the appropriate bit in the PmiIntr register. Others can only be cleared by sifting through the causality hierarchy to find the origin.

In the event that the RC signals that a legacy interrupt has been asserted, this assertion will not be readable via the CSW until all write commands that preceeded the interrupt message have been completed by the CCW.

13.6.3.6 CSI Exception Handling

Errors conditions can arise in a number of places in the CSI:

Data with Bad ECC from CSW If data with an ECC error arrives from the CSW, the error details will be logged in the PmiCsiEccErr register (section 13.13.10) and a bit set in the PmiIntr register (section 13.13.2). The transaction will be completed regardless of whether the error was of a single bit or double bit nature.

Out of range address from CSW If a request arrives whose address does not pertain to any of the subfunctions within the CSI, the error details will be logged in the PmiCsiAddrErr register (section 13.13.11) and a bit set in the PmiIntr register (section 13.13.2). The CSW protocol will be completed, meaning that read data with all ones will be returned for a read and a RDIO will be issued for a write with the subsequently returned data being discarded.

64 bit DBI access request The DBI port to the RC has a 32 bit data path. If an access of more than 32 bits is requested, the error details will be logged in the PmiCsiDbiErr register (section 13.13.12) and a bit set in the PmiIntr register (section 13.13.2). The CSW protocol will be completed, meaning that read data with all ones will be returned for a read and a RDIO will be issued for a write with the subsequently returned data being discarded.

Wishbone Timeout If an access to a wishbone component (the UART or I2C) times out, , the error details will be logged in the PmiCsiWtoErr register (section 13.13.13) and a bit set in the PmiIntr register (section 13.13.2). The CSW protocol will be completed, meaning that read data with all ones will be returned for a read. (The protocol for a write had already completed prior to the transaction to the wishbone being started and hence before the timeout.)

13.6.4 The Command/Address Multiplexer CMX

The Command/Address multiplexer takes command inputs from each of the command processing units (REQ, CMP, CSI, and CIN). Requests from the CMP are given priority over the other three, who are selected on a LRU basis. The command processing units can only present requests one at a time and move onto a new request only when given a grant.

The CMX also buffers the Command/Address from the CSW headed to the CMP, REQ or CSI. It parses the address to determine the target of the incoming command. There is no throttling mechanism for incoming requests from the CSW, so they are parsed and sent to FIFOs within each of the target units.

13.6.5 The Data Multiplexer DMX

The data multiplexer accepts inputs from each of the data sourcing units (REQ, CMP, and CSI). Requests from the CMP are given priority over the other two, who are selected on a LRU basis. The data sourcing units can only present requests one at a time and move onto a new request only when given a grant. In the case of requests from the REQ and CSI, the data can only be up to 64 bits in length and hence is accepted at the time of the grant. Data from the CMP is 64B in length. At the time of a request from the CMP, the DMX will immediately issue a grant as long it is not busy servicing another request. The data from the CMP is then streamed into the DMX in preparation for streaming onto the CSW data lines as soon as the CSW grant is received. This puts the outbound data right next to the CSW and allows the buffer within the CMP to be freed for the assembly of the next transaction. The DMX generates ECC for all data headed to the CSW. The PmiFrcEccErr register (section 13.13.9) allows the purposeful corruption of the data headed out to the CSW. The DMX also buffers the data transactions from the CSW.

13.7 Valid CSW Operations

The PMI both accepts commands/data from the CSW and sends commands/data to the CSW. The following enumerates the sequence of events that are permissible in interacting with the PMI. The nomenclature used is that "PMI:BWT(COH)" means that a BWT command was sent by the PMI to the COH via the CSW.

CSW:RDIO -> PMI:DATA
CSW:WTIO -> PMI:RDIO -> CSW:DATA
PMI:BWT(COH) -> CSW:BWTGO(COH) -> PMI:DATA(COH)
PMI:BWT(COH) -> CSW:BWTGO(PX) -> PMI:DATA(PX)
PMI:BWT(COH) -> CSW:BWTNOHIT -> PMI:DATA(COH)
PMI:BWT(COH) -> CSW:PRBINV -> PMI:DATA(COH)
PMI:RDEX(COH) -> CSW:DATA(COH)
PMI:RDEX(COH) -> CSW:DATA(PX) -> PMI:PRBDONE(COH)
PMI:RDEX(COH) -> CSW:PRBNOHIT -> PMI:RDEXR(COH) -> CSW:DATA(COH)
PMI:WINV(COH) -> PMI:DATA(COH)
PMI:BRD(COH) -> CSW:DATA(COH)
PMI:BRD(COH) -> CSW:DATA(PX) -> PMI:PRBDONE(COH)
PMI:BRD(COH) -> CSW:PRBNOHIT -> PMI:BRDR(COH) -> CSW:DATA(COH)
CSW:PRBWIN(PX) -> PMI:PRBNOHIT
CSW:PRBBWT(PX) -> PMI:BWTNOHIT
CSW:PRBBRD(PX) -> PMI:PRBNOHIT
CSW:PRBSHR(PX) -> PMI:PRBNOHIT
PMI:INTR -> CSW:DONE

13.8 Valid PCI Operations

Coming from an endpoint, the RC and PMI will only accept completions, MemWrites, MemReads, vendor messages and MSIs (which look just like MemWrites). A core within the ICE9 can initiate a MemWrite, MemRead, IO Write, IO Read, Config Write or Config Read transaction headed to a downstream device. It can also cause certain status messages to be sent, as specified in the register definitions below and in the RC specification.

All Config and IO transactions use 32 bit addressing and data. A MemWrite or MemRead can use 32 or 64 bit addressing and up to 64 bits of data. A Mem command to an address with Addr[63:32] = 0x8 will result in 3 DW

header (32 bit address) being sent. Mem commands with those same bits set to 0x9, 0xA or 0xB will cause a 4 DW header (64 bit address)to be sent. A practical consequence of this is that 32 bit endpoints will use up some of the bottom 4GB of the main memory allocation.

13.9 Ordering Rules

Before stating the ordering rules, it would be good to define a couple terms. An “inbound” transaction is one originating at an endpoint and heading to the ICE9. An “outbound” transaction is one originating within the ICE9 and heading to an endpoint.

Inbound transactions can only be posted operations (memory writes, message signalled interrupts (MSIs) or vendor messages), memory reads or completions. Posted operations are handled in order; a posted operation can not pass another posted operation. The exception to this, in the case of the ICE9, is that vendor messages are handled at presentation from the PRC to the PMI, whereas the other two types of posted operations are stuck in a queue and handled when they get to the top of the queue. Posted operations can pass memory reads and completions. Memory reads and completions are also handled in order of presention, but neither are handled before any posted operation that preceded it. Completions can, however, pass memory reads.

Outbound transactions can be posted (memory writes or vendor messages), non-posted (config reads/writes, IO reads/writes or memory reads) or completions. Similar rules as above apply to the outbound transactions. Posted operations occur in order except that vendor messages can pass memory writes. Non-posted operations occur in order, as do completions. Completions can pass non-posted operations, but can not pass posted operations.

For the puposes of the ICE9, the “timestamp” of an operation is not when it first comes across the CSW, but when it gets to the top of the REQ queue and, if needed, the associated data has been retrieved from the originating processor.

13.10 Auxiary PCI Signals

There are a number of signals needed to control the PCI Express module or card.

13.10.1 PERST# output

PCIe express module or card fundamental reset. Active low on the PCB. Resets the PCIe card or express module attached to the ICE9 when asserted. The logic is PERST# = ˜(ResetCard | MPWRGD#). The ResetCard signal is bit 11 in the Core Control Register (section 13.13.1). Drives PERST# on cards and MRST# on express modules.

13.10.2 MPWRGD# input

PCIe express module power good. Active low on PCB. See PERST# for usage. On CPU modules, which support PCIe express modules, MPWRGD# is pulled up on the PCB. Therefore, MPWRGD# is deasserted by default; an express module must drive it low to assert it, and PERST# cannot be deasserted until it does so. On development boards, which support PCIe cards, MPWRGD# is pulled down on the PCB; therefore, it is always asserted. This is necessary since PCIe cards don’t support MPWRGD# and PERST# could never be deasserted otherwise.

13.10.3 PWRFLT# input

PCIe express module power fault. Active low on PCB. When asserted a 1 should appear in Slot Status Register[1] (Power Fault Detected). This is probably meant to be a sticky bit since PWRFLT can be transient. On CPU modules and development boards PWRFLT# is pulled up on the PCB. Therefore, it is deasserted by default; an express module must drive it low to assert it. PCIe cards don’t support this signal, so it is never asserted on development boards.

13.10.4 PWREN# output

PCIe express module power enable. Active low on PCB. Driven by Slot Control Register[10] (Power Controller Control).

13.10.5 PRSNT# input

PCIe express module or card present. Active low on PCB. When asserted a 1 should appear in Slot Status Register[6] (Presence Detect State), otherwise a 0. Presence Detected Changed presumably has to get set when PRSNT changes state, and is a sticky bit.

13.10.6 ATNLED output

PCIe express module attention LED. A state machine controls this output, which can be on, off, or blinking. The output behavior is defined by Slot Control Register[7:6]). If blinking, the on or off time of the 50% duty cycle signal is defined by the LED Blink Rate Register (section 13.13.4). This register gives the high or low time in clock cycles; the frequency should be 1-2Hz.

13.10.7 PWRLED output

PCIe express module power LED. A state machine controls this output, which can be on, off, or blinking. The output behavior is defined by Slot Control Register[9:8]). If blinking, the on or off time of the 50% duty cycle signal is defined by the LED Blink Rate Register (section 13.13.4). This register gives the high or low time in clock cycles; the frequency should be 1-2Hz.

13.11 Definitions

Package

chip_pci_spec

13.11.1 PCI Type Enumerations

Enum

PciType

Constant	Mnemonic	Definition
5'h0	MRW	Memory Reads and Writes
5'h1	MRLK	Reserved. (Memory Read Request-Locked)
5'h2	IORW	IO Reads and Writes
5'h4	CFG0RW	Config Type 0 Reads and Writes
5'h5	CFG1RW	Config Type 1 Reads and Writes
5'ha	CPL	Completions
5'hb	CPLLK	Completions for Locked Memory Reads

13.11.2 PCI Format Enumerations

Enum

PciFmt

Constant	Mnemonic	Definition
2'h0	NODAT3DWH	3 DW header without data
2'h1	NODAT4DWH	4 DW header without data
2'h2	DAT3DWH	3 DW header with data
2'h3	DAT4DWH	4 DW header with data

13.11.3 PCI Completion Status Enumerations

Enum

PciCplStat

Constant	Mnemonic	Definition
3'h0	SC	Successful Completion
3'h1	UR	Unsupported Request
3'h2	CR	Configuration Request Retry Status
3'h4	CA	Completer Abort

13.11.4 PCI Completion State Machine State Enumerations

Enum

PciCmpSm

Constant	Mnemonic	Definition
2'h0	IDLE	Idle state
2'h1	WAIT	Wait for data to accumulate
2'h2	STREAM	Stream data out to RC

13.11.5 PCI Block Write State Machine State Enumerations

Enum

PciBwtSm

Constant	Mnemonic	Definition
4'h0	IDLE	Idle state
4'h1	RDEXCMD	Sending RDEX command
4'h2	RDEXDATA	Receiving RDEX data
4'h3	PRBDONCMD	Sending PRBDONE command
4'h4	WINVCMD	Sending WINV command
4'h5	WINVDATA	Sending WINV data
4'h6	BWTCMD	Sending BWT command
4'h7	BWTDATA	Sending BWT data
4'h8	INTRCMD	Sending an INTR command
4'h9	INTRDONE	Waiting on the DONE in response to the INTR

13.11.6 PCI Block Read State Machine State Enumerations

Enum

PciBrdSm

Constant	Mnemonic	Definition
2'h0	IDLE	Idle state
2'h1	BRDCMD	Sending BRD command
2'h2	BRDRCMD	Sending a BRDR command
2'h3	PRBDONCMD	Sending PRBDONE command

13.11.7 PMI Request Result Enumerations

Enum

PmiReqRes

Constant	Mnemonic	Definition
4'h0	NODAT	Successful Completion without Data
4'h1	DAT32	Successful Completion with 32-bit Data
4'h2	DAT64	Successful Completion with 64-bit Data
4'h3	UNSUPPORTED	Unsupported Request
4'h4	POISONED	Poisoned
4'h5	BADECRC	ECRC error detected by Root-Complex
4'h6	BADLENGTH	Bad TLP length received
4'h7	DLLPABORT	DLLP Abort asserted by Root-Complex
4'h8	TLPABORT	TLP Abort asserted by Root-Complex
4'h9	TIMEOUT	Request timed out
4'ha	RETRY	Config retry
4'hb	ABORT	Completer Abort

13.11.8 Pmi Events

The following events are trackable by SCB statistical event counting.

Enum

PmiScbEvent

Attributes

-descfunc

Constant	Mnemonic	Definition
8'h00	CYCLES	Core clock cycles. Always counts.
8'h01	CONFIGW_OUT	Number of outbound PCI Config Write transactions.
8'h02	CONFIGR_OUT	Number of outbound PCI Config Read transactions.
8'h03	PCI_IOW_OUT	Number of outbound PCI IO Write transactions.
8'h04	PCI_IOR_OUT	Number of outbound PCI IO Read transactions.
8'h05	MEMW32_OUT	Number of outbound PCI Memory Writes with 32 bit data.
8'h06	MEMW64_OUT	Number of outbound PCI Memory Writes with 64 bit data.
8'h07	MEMR32_OUT	Number of outbound PCI Memory Reads with 32 bit data.
8'h08	MEMR64_OUT	Number of outbound PCI Memory Reads with 64 bit data.
8'h09-8'h0f		Reserved.
8'h10	MEMWA64_IN	Number of inbound aligned memory writes with data of 64B or less.
8'h11	MEMWA128_IN	Number of inbound aligned memory writes with data of 128B or less.
8'h12	MEMWA256_IN	Number of inbound aligned memory writes with data of 256B or less.
8'h13	MEMWA512_IN	Number of inbound aligned memory writes with data of 512B or less.
8'h14	MEMWU64_IN	Number of inbound unaligned memory writes with data of 64B or less.
8'h15	MEMWU128_IN	Number of inbound unaligned memory writes with data of 128B or less.
8'h16	MEMWU256_IN	Number of inbound unaligned memory writes with data of 256B or less.
8'h17	MEMWU512_IN	Number of inbound unaligned memory writes with data of 512B or less.
8'h18-8'h1f		Reserved.

8'h20	MEMRA64_IN	Number of inbound aligned memory reads with data of 64B or less.
8'h21	MEMRA128_IN	Number of inbound aligned memory reads with data of 128B or less.
8'h22	MEMRA256_IN	Number of inbound aligned memory reads with data of 256B or less.
8'h23	MEMRA512_IN	Number of inbound aligned memory reads with data of 512B or less.
8'h24	MEMRU64_IN	Number of inbound unaligned memory reads with data of 64B or less.
8'h25	MEMRU128_IN	Number of inbound unaligned memory reads with data of 128B or less.
8'h26	MEMRU256_IN	Number of inbound unaligned memory reads with data of 256B or less.
8'h27	MEMRU512_IN	Number of inbound unaligned memory reads with data of 512B or less.
8'h28-8'hff		Reserved.

13.12 PCI Express Root Complex Registers

All of the registers in this section are within the Synopsys Root Complex. The details of these registers were taken from the document supplied by Synopsys.

13.12.1 Device/Vendor ID Register

Description

Register

R_PcieId

Attributes

-kernel

Address

0xE_9800_0000

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:16	**DeviceID**	**RW**	**1**		**Device ID.**
15:0	**VendorID**	**RW**	**0x19B2**		**Vendor ID. Assigned by PCI-SIG.**

13.12.2 Command and Status Register

Description

Register

R_PcieCmdStat

Attributes

-kernel -writeonemixed

Address

0xE_9800_0004

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31	DetParErr	RW1C	0		1 = forwarding an outbound Poisoned TLP (bit is set regardless of ParErrEn).
30	SigSysErr	RW1C	0		Set when RC generates ERR_(NON)FATAL message and SerrEn = 1 in the Command Register (bit 40 in this register)
29	RcvdMstrAbrt	RW1C	0		Set when primary side of RC receives UR Completion Status for Request.
28	RcvdTgtAbrt	RW1C	0		Set when primary side of RC receives CA Completion Status for Request.
27	SigTgtAbrt	RW1C	0		Set when RC sends CA Completion Status for Request.
26:25		R	0		Reserved
24	MstrDatParErr	RW1C	0		1 = received (and forwarding) an inbound Poisoned TLP (and Parity Error Response bit - bit 38 - in Command portion of this Register is set).
23		R	0		Reserved.
22		R	0		Reserved.
21		R	0		Reserved
20	CapList	R	1		Indicates presence of extended Capabilities List.
19	IntStatus	R	0		Indicates pending INTx Message. Irrelevant for RC.
18:11			0		Reserved
10	IntDis	RW	0		Disables INTx interrupts from being sent.
9		R	0		Reserved
8	SerrEn	RW	0		(Non)fatal error messages (from Endpoint) reported if this bit is 1 (or if another bit in Device Control Register is set). This reporting takes the form of updating Root Control register and/or Root Error Command register, and logging in the Root Error Status register and Error Source ID register.
7		R	0		Reserved
6	ParErrResp	RW	0		Parity Error Response
5:3		R	0		Reserved
2	BusMstrEn	RW	0		Bus Master Enable. 0 = treat incoming MEM/IO requests as Unsupported Request.
1	MemDecEn	RW	0		1 = Allow MEM accesses from Endpoint
0	IoDecEn	RW	0		1 = Allow IO accesses from Endpoint

13.12.3 RevID, Class Code Register

Description

Register

R_PcieRevId

Address

0xE_9800_0008

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8	ClassCode	RW	0x060400		*Class Code.*
7:0	RevID	RW	1		*Revision ID.*

13.12.4 Cache Line Size, BIST etc register

Description

Unsure what the Revision ID and Class Code is for this chip. The Cache Line Size is irrelevant for PCI Express functionality. Master Latency Timer Register is hardwired to 0. Need to find the details of Header Type and BIST fields.

Register

R_PcieCcMisc

Address

0xE_9800_000C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:24	Bist	R	0		Not supported by RC Core.
23	MfDev	RW	0		Multi Function Device
22:16	HdrTyp	R	1		Config Header Format
15:8	MstrLatTim	R	0		Hardwired to 0.
7:0	CacLinSiz	RW	0		*System Cache Line Size.* Irrelevant for us.

13.12.5 Base Address Register 0

Description

The Base Address Registers specify the windows for Memory and IO access from the endpoint. For our Root Port, we have no need for this and so will keep it at 0.

Register

R_PcieBar0

Address

0xE_9800_0010

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	BaseAddr	R	0x4		[31:4} are 0. [3:0] indicate non-prefetchable (0), 64 bit (10), memory (0).

13.12.6 Base Address Register 1

Description

The Base Address Registers specify the windows for Memory and IO access from the endpoint. For our Root Port, we have no need for this and so will keep it at 0.

Register

R_PcieBar1

Address

0xE_9800_0014

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	BaseAddr	R	0		Hardwired to 0.

13.12.7 Bus Number Register

Description

The Primary Bus number for a Root Complex is 0. The Secondary Bus number is 1. The Subordinate Bus number can be any number from 1 (indicating an Endpoint connection) to a number greater than 1 (indicating a Switch connection).

Register

R_PcieBusNum

Address

0xE_9800_0018

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:24	SecLatTim	R	0		Hardwired to 0.
23:16	SubBusNum	RW	0		Subordinate Bus Number
15:8	SecBusNum	RW	0		Secondary Bus Number
7:0	PriBusNum	RW	0		Primary Bus Number

13.12.8 I/O Base/Limit, and Secondary Status Register

Description

Register

R_PcieSecStat

Attributes

-writeonemixed

Address

0xE_9800_001C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31	DetParErr	RW1C	0		1 = received Poisoned TLP in inbound direction (regardless of ParErrResp bit value in Bridge Control register).

Bit	Mnemonic	Access	Reset	Type	Definition
30	RcvSysErr	RW1C	0		1 = Received incoming ERR_(NON)FATAL message (Errata doc on Page 104 says that this bit is not dependent on SERR Enable bit in Bridge Control Register).
29	RcvMstrAbrt	RW1C	0		Set when RC receives UR Completion Status for outbound Request.
28	RcvTgtAbrt	RW1C	0		Set when RC receives CA Completion Status for outbound Request.
27	SigTgtAbrt	RW1C	0		Set when RC sends CA Completion Status for inbound Request.
26:25		R	0		Reserved
24	MstrDataParErr	RW1C	0		1 = sent an outbound Poisoned TLP (and Parity Error response bit - bit 0 - in Bridge Control Register is set). If ParErrResp bit in Bridge Control register is 0, this bit is always 0.
23:16		R	0		Reserved
15:12	IoLimit74	RW	0		IO Limit Register Value (alongwith implicit zeroes in lower 12 bits, provides end/limit of address space of outbound IO transactions in 64KB address space).
11:8	IoLimit30	R	0x1		0 = 16-bit IO address decode (64KB space). 1 = 32-bit IO address decode (4GB space). Value in IO Upper Limit Register valid if this value is 1. Values 0x2 through 0xF are reserved.
7:4	IoBase74	RW	0		IO Base register value (alongwith implicit zeroes in lower 12 bits, provides start address space of outbound IO transactions in 64KB address space).
3:0	IoBase30	R	0x1		0 = 16-bit IO address decode (64KB space). 1 = 32-bit IO address decode (4GB space). Value in IO Upper Base Register valid if this value is 1. Values 0x2 through 0xF are reserved.

13.12.9 Non-Prefetchable Memory Base and Limit Register

Description

These registers define the start and end address range for valid outbound Memory transactions.

Register

R_PcieMemBase

Address

0xE_9800_0020

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:20	MemLmt	RW	0		End Address of Memory range. Upper 12 bits of implicit 32-bit range (lower 20 bits are assumed 0xFFFFF).
19:16		R	0		Reserved
15:4	MemBase	RW	0		Start Address of Memory range. Upper 12 bits of implicit 32-bit range (lower 20 bits are assumed 0).
3:0		R	0		Reserved

13.12.10 Prefetchable Memory Base and Limit Register

Description

These registers define the start and end address range for valid outbound Memory transactions.

Register

R_PciePreMemBase

Address

0xE_9800_0024

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:20	PMemLmt	RW	0		Prefetchable Memory Limit
19:17		R	0		Reserved
16	Addr64L	RWS	1		64 bit addressing if a one
15:4	PMemBas	RW	0		Prefetchable Memory Base
3:1		R	0		Reserved
0	Addr64B	RWS	1		64 bit addressing if a one

13.12.11 Prefetchable Memory Upper Base Register

Description

These are the upper bits of prefetchable memory base.

Register

R_PciePreBaseUpper

Address

0xE_9800_0028

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	PMemBU	RW	0		Prefetch Memory Base Upper register

13.12.12 Prefetchable Memory Upper Limit Register

Description

These are the upper bits of prefetchable memory limit.

Register

R_PciePreLimitUpper

Address

0xE_9800_002C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	PMemLU	RW	0		Prefetch Memory Limit Upper Register

13.12.13 I/O Base and Limit Upper Register

Description

These registers define the Upper Base and Limit range for outbound IO space if that space is 32-bits wide.

Register

R_PcieIOUpperBaseLimit

Address

0xE_9800_0030

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:16	IoLimitU	RW	0		Upper 16 bits of IO Limit register (only valid if outbound IO space is in 4GB space rather than 64KB space).
15:0	IoBaseU	RW	0		Upper 16 bits of IO Base register (only valid if outbound IO space is in 4GB space rather than 64KB space).

13.12.14 Capability Pointer Register

Description

Register

R_PcieCapabilityPtr

Address

0xE_9800_0034

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8		R	0		Reserved.
7:0	CapPtr	RW	0x40		Capability Pointer. Points to (contains the offset to) register set associated with the next Capability.

13.12.15 Expansion ROM Register

Description

We do not support an Expansion ROM within the Root Complex bridge.

Register

R_PcieExpRom

Address

0xE_9800_0038

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:4	Address	R	0		Reserved. Expansion ROM not supported.
3:1		R	0		Reserved
0	Enable	R	0		Expansion ROM enable

13.12.16 Bridge Control Register

Description

Register

R_PcieBrgCtrl

Address

0xE_9800_003C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:23		R	0		Reserved
22	SecBusRst	RW	0		1 = Triggers Hot Reset on PCI-E link.
21	MstrAbort	R	0		Not applicable
20	VGA16	RW	0		VGA 16 bit decode
19	VGAEn	RW	0		VGA Enable
18	ISAEn	RW	0		ISA Enable
17	SerrEn	RW	0		1 = Allows forwarding of received ERR_{COR, NONFATAL, FATAL} error messages to primary side of Bridge. The SerrEn bit in the Command Register controls reporting of these forwarded messages to the Root Complex.
16	ParErrResp	RW	0		1 = Enable Master Data Parity Error status bit in both primary and secondary status registers.
15:8	IntPin	RW	0x1		Interrupt Pin register. Irrelevant for Root Complex.
7:0	IntLine	RW	0xff		Interrupt Line register. Irrelevant for Root Complex.

13.12.17 PCI Power Management Capabilities Register

Description

Register

R_PciePMCap

Address

0xE_9800_0040

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31		R	0		Reserved
30:27	PMESup	RWS	0xb		Bits 30, 28, 27 set to 1 for Root Port to indicate in which states it will forward received PME Messages to the Root Complex.
26	D2Sup	RW	0		D2 Support

Bit	Mnemonic	Access	Reset	Type	Definition
31		R	0		Reserved
25	D1Sup	RW	1		D1 Support
24:22	AuxCur	RW	0x7		Auxiliary current
21	SpecInit	RW	0		Device Specific Initialization
20:19		R	0		Reserved
18:16	CapVer	RW	2		Capability Version (as mandated by PCI-SIG)
15:8	NxtCapPtr	RW	0x50		Offset to next PCI capability structure
7:0	CapId	R	0x01		ID indicating PCI Express Capability Structure.

13.12.18 PCI Power Management Control Register

Description

Register

R_PciePMCtrl

Attributes

-writeonemixed

Address

0xE_9800_0044

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:24		R	0		Reserved
23:22		R	0		Reserved
21:16		R	0		Reserved
15	PMESt	RW1C	0		Root Complex will not set this bit.
14:9		R	0		Reserved
8	PMEEn	RW	0		Since Root Complex never sends PME Message, this bit can be hardwired to 0.
7:0		R	0		Reserved

13.12.19 MSI Capabilities Register

Description

Register

R_PcieMSICap

Address

0xE_9800_0050

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:24		R	0		Reserved.
23	MSI64En	RWS	1		64-bit Address Capable
22:20	MultiMSIEn	RW	0		Multiple Message Enabled
19:17	MultiMSICap	RW	0		Multiple Message Capable (writable through DBI)
16	MsiEn	RW	0		MSI Enabled (when set, INTx must be disabled)

Bit	Mnemonic	Access	Reset	Type	Definition
15:8	NxtCapPtr	RW	0x70		Offset to next PCI capability structure
7:0	CapId	R	0x05		ID indicating MSI Capability.

13.12.20 MSI Address Register

Description

Contains the MSI Lower 32-bit address (only upper 30 of these 32 bits are writable).

Register

R_PcieMSIAddr

Address

0xE_9800_0054

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:2	MSIAddrL	RW	0		MSI Lower 32-bit Address
1:0		R	0		Reserved.

13.12.21 MSI Upper Address/Data Register

Description

Bits 31:0 in this register contain the MSI Upper Address Register, if MSI64En = 1. Otherwise, it contains the MSI Data Register.

Register

R_PcieMSIUpper

Address

0xE_9800_0058

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	MSIAddrH	RW	0		Upper 32-bit Address (or MSI Data register if MSI64En=0)

13.12.22 MSI Data Register

Description

Contains the MSI Data register is MSI64En = 1.

Register

R_PcieMSIData

Address

0xE_9800_005C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:16		R	0		Reserved.
15:0	MSIData	RW	0		MSI Data (if MSI64En=1)

13.12.23 PCI Express Capabilities Register 0

Description

Register

R_PcieCap0

Address

0xE_9800_0070

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:30		R	0		Reserved
29:25	IntMsgNum	RW	0		Interrupt Message Number
24	SlotImp	RW	1		1 = Link connected to a Slot. Hardware initialized to a 1.
23:20	PortType	R	0x4		Device/Port Type. 4 = Root Port of PCIE Root Complex
19:16	CapVer	R	1		Capability Version (as mandated by PCI-SIG)
15:8	NxtCapPtr	RW	0x0		Offset to next PCI capability structure
7:0	CapId	R	0x10		ID indicating PCI Express Capability Structure.

13.12.24 PCI Express Capabilities Register 1

Description

Register

R_PcieCap1

Address

0xE_9800_0074

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:16		R	0		Reserved
15	RBErrRep	RW	1		Role Based Error Reporting
14:12		R	0		Reserved
11:9	L1Lat	RW	1		Endpoint Acceptable L1 latency
8:6	L0sLat	RW	1		Endpoint Acceptable L0s latency
5	ExtTag	RW	0		Only 5-bit Tag field supported.
4:3	PhanFunc	RW	0		No Phantom Functions supported.
2:0	MaxPaySiz	RW	0x2		Max Payload Size Supported. 2 = 512 bytes.

13.12.25 Device Control/Status Register

Description

Register

R_PcieDevCtlStat

Attributes

-writeonemixed

Address

0xE_9800_0078

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:22		R	0		Reserved
21	TrnPend	R	0		1 = Outbound non-posted transactions pending (ie. have not completed or have not been terminated by the Completion Timeout mechanism)
20	AuxPwrDet	R	0		0 = No Aux Power Detected
19	URDet	RW1C	0		1 = Unsupported Request Detected. Independent of any control or mask setting.
18	FatErrDet	RW1C	0		1 = Fatal Error Detected. Independent of any control or mask setting.
17	NFErrDet	RW1C	0		1 = Non-Fatal Error Detected. Independent of any control or mask setting.
16	CorErrDet	RW1C	0		1 = Correctable Error Detected. Independent of any control or mask setting.
15		R	0		Reserved
14:12	MaxRdReq	R	0x2		Maximum permissible inbound read request size.
11	NoSnpEn	RW	0		Always 0. We do not enable “No Snoop”.
10	AuxPowEn	RW	0		Enable Aux Power.
9:8		R	0		Reserved
7:5	MaxPaySiz	RW	0		0 = 128 bytes, 1 = 256 bytes, 2 = 512 bytes.
4	EnRlxOrd	RW	1		Always 0. We do not enable Relaxed Ordering.
3	UrRepEn	RW	0		1 = enables reporting of Unsupported Request (ie. inbound packet encounters a UR which needs to be reported to the host)
2	FatErrEn	RW	0		1 = Enables reporting of fatal errors (equivalent of enabling ERR_FATAL messages for a Root Port)
1	NFErrEn	RW	0		1 = Enables reporting of non-fatal errors (equivalent of enabling ERR_NONFATAL messages for a Root Port)
0	ErrCorrEn	RW	0		1 = Enables reporting of correctable errors to the host (equivalent of enabling ERR_COR messages for a Root Port)

13.12.26 Link Capabilities Register

Description

Register

R_PcieLnkCap

Address

0xE_9800_007C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:24	PortNum	RW	0		Port Number for the PCI Express Link.
23:21		R	0		Reserved
20	DLLEn	R	1		Data Link Layer Active Reporting Capable
19	SurDwn	R	0		Surprise Down Error Reporting Capable
18	ClkPmCap	RW	0		Clock Power Management
17:15	L1ExLat	RW	0x6		L1 Exit Latency. Irrelevant for us since we do not support L1.
14:12	L0sExLat	RW	0x3		L0s Exit Latency.
11:10	ASPM	RW	0x3		Active Link Pm Support
9:4	MaxLnkWth	RW	0x8		Max Link Width. 8 lanes in our case.
3:0	MaxLnkSpd	RW	0x1		1 = 2.5Gb/s Link. All other encodings are reserved.

13.12.27 Link Control/Status Register

Description

Register

R_PcieLnkCtl

Address

0xE_9800_0080

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:30		R	0		Reserved
29	DLLayerAct	R	0		1 = Data Link Layer Active
28	SltClkCfg	RW	1		1 = Component uses same reference clock as on the connector. Initialized by hardware to correct value.
27	LnkInTrn	R	0		1 = Link Training in progress. Should be set to 0 by hardware after successful training to the L0 state.
26	TrainErr	R	0		1 = Link Training Error occurred. Should be set to 0 by hardware after successful training to the L0 state.
25:20	NegLnkWth	R	1		Negotiated link width. We should see values of 1, 2, 4, or 8.
19:16	LnkSpd	R	0x1		1 = 2.5Gb/s. All other encodings are reserved.
15:8		R	0		Reserved
7	ExtSync	RW	0		1 = Forces extra FTS ordered sets when transitioning from low power states to L0.
6	ComClkCfg	RW	0		1 = Common Reference Clock at both sides of the Link. 0 = Asynchronous clocks at both sides of the link.
5	LnkRetrain	R	0		1 = Initiate Link retraining via the Recovery State. Reads always return 0.
4	LnkDis	RW	0		1 = disable the Link.
3	RCB	RW	0		Read Completion Boundary. 0 = 64 bytes.
2		R	0		Reserved
1:0	ASPMCtl	RW	0		1 = L0s Entry Enabled. Can be disabled by writing 0x0.

13.12.28 Slot Capabilities Register

Description

Register

R_PcieSltCap

Address

0xE_9800_0084

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:19	PhySltNum	RW	0x0		Physical Slot Number. I believe this should be 0 for a Root Port.
18	SltNoCCSup	RW	0		Slot No Command Complete Support
17	SltEmlPrsnt	RW	0		Slot Electromechanical Interlock Present
16:15	SltPwrScl	RW	0		Slot Power Limit Scale. Writes to this register cause Port to send Set_Slot_Power_Limit Message.
14:7	SltPwrLmt	RW	0xf		Slot Power Limit Value. Writes to this register cause Port to send Set_Slot_Power_Limit Message.
6:0	SlotCap	RW	0x7a		These 7 bits in the Slot Capabilities register are all hardware initialized to some value.

13.12.29 Slot Control/Status Register

Description

Register

R_PcieSltCtl

Attributes

-kernel -writeonemixed

Address

0xE_9800_0088

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:23		R	0		Reserved.
22	PrnDetSt	R	0		1 = Indicates presence of card in slot. 0 = Slot Empty.
21	MRLSenSt	R	0		MRL Sensor State. 0 = MRL Closed. 1 = MRL Open.
20	CmdCpl	RW1C	0		1 = Hot Plug Controller completes an issued command.
19	PrnDetChg	RW1C	pins		1 = Presence Detect change is detected.
18	MRLSenChg	RW1C	0		1 = MRL Sensor State Change is detected.
17	PwrFltDet	RW1C	0		1 = Power Controller detects power fault in this slot.
16	AttButPrs	RW1C	0		1 = Attention Button is Pressed.
15:11		R	0		Reserved and Preserved.
10	PwrCtlCtl	RW	0		1 = Power applied to the slot is Off. 0 = Power applied is On.

Bit	Mnemonic	Access	Reset	Type	Definition
9:8	PwrIndCtl	RW	0x3		Non-zero writes to this register set these bits as well as send the appropriate PWR_INDICATOR_{ON,OFF,BLINK} Message.
7:6	AttIndCtl	RW	0x3		Non-zero writes to this register set these bits as well as send the appropriate ATTN_INDICATOR_{ON,OFF,BLINK} Message.
5	HotPlugEn	RW	0		1 = Enable Hot-Plug interrupt generation for enabled Hot-Plug events.
4	CmdCplEn	RW	0		1 = Enable Hot-Plug interrupt generation for Command Completion by Hot Plug Controller.
3	PrnDetEn	RW	0		1 = Enable Hot-Plug interrupt generation for presence detect changed event.
2	MRLSenEn	RW	0		1 = Enable Hot-Plug interrupt generation for MRL Sensor Changed event.
1	PwrFltEn	RW	0		1 = Enable Hot-Plug interrupt generation for power fault event.
0	AttButEn	RW	0		1 = Enable Hot-Plug interrupt generation for Attention Button Pressed event.

13.12.30 Root Control Register

Description

Register

R_PcieRootCtl

Address

0xE_9800_008C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:4		R	0		Reserved
3	PMEIntEn	RW	0		1 = Root Port should generate interrupt if PME Status register bit is set indicating receipt of PME Message. If PME Status bit is already set when this bit is enabled, interrupt should be generated. (Errata doc Page 110 says that the Root Port should generate interrupt wire only when Interrupt Disable bit in Command Register is 0 in addition to the above 2 bits being set).
2	FatErrEn	RW	0		1 = RC should generate system error if Fatal Error reported by the Root Port or by devices in its hierarchy. This should not happen if SerrEn bit in Command register = 0 (based on Errata doc Page 107).
1	NFErrEn	RW	0		1 = RC should generate system error if NonFatal Error reported by the Root Port or by devices in its hierarchy.This should not happen if SerrEn bit in Command register = 0 (based on Errata doc Page 107).
0	CorrErrEn	RW	0		1 = RC should generate system error if Correctable Error reported by the Root Port or by devices in its hierarchy.This should not happen if SerrEn bit in Command register = 0 (based on Errata doc Page 107).

13.12.31 Root Status Register

Description

The Root Status Register are described here.

Register

R_PcieRootStatus

Address

0xE_9800_0090

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:18		R	0		Reserved.
17	PMEPend	R	0		1 = Another PME is pending when PME Status bit is set. When PME Status is cleared by software, pending PME will cause PME Status to be set again with the updated Req ID. Process will continue until no more PMEs are pending.
16	PMEStat	RW1C	0		1 = PME was asserted by requester in bits 15:0.
15:0	PMEReqId	R	0		Indicates PCI Requester ID of the last PME requester.

13.12.32 Advanced Error Reporting Enhanced Capability Header Register

Description

Register

R_PcieAdvErrCapHdr

Address

0xE_9800_0100

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:20	NxtCapOff	R	0x0		Next Capability Offset (relative to address 0 of config space)
19:16	CapVer	R	0x1		Capability Version. Assigned by PCI-SIG.
15:0	ExtCapId	R	0x1		PCI Express Extended Capability ID. Assigned by PCI-SIG.

13.12.33 Advanced Error Reporting Uncorrectable Error Status Register

Description

Bits in this register are sticky and report the error status of individual error sources.

Register

R_PcieUCorrErr

Address

0xE_9800_0104

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:21		R	0		Reserved
20	URErrSt	RW1C	0		Unsupported Request Error Status
19	ECRCErrSt	RW1C	0		ECRC Error Status
18	MlfTLPSt	RW1C	0		Malformed TLP Status
17	RcvOvfSt	RW1C	0		Receiver Overflow Status
16	UnxCplSt	RW1C	0		Unexpected Completion Status
15	CplAbrtSt	RW1C	0		Completer Abort
14	CplTOSt	RW1C	0		Completion Timeout Status
13	FCErrSt	RW1C	0		Flow Control Protocol Error Status
12	PsnSt	RW1C	0		Poisoned TLP Status
11:5		R	0		Reserved.
4	DLErrSt	RW1C	0		Data Link Protocol Error Status
3:1		R	0		Reserved.
0	TrnErrSt	RW1C	0		Training Error Status (default undefined for Rev 1.1)

13.12.34 Uncorrectable Error Mask Register

Description

All unreserved bits in these registers are sticky.

Register

R_PcieUncErrMsk

Address

0xE_9800_0108

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:21		R	0		Reserved
20	URErrMsk	RWS	0		Unsupported Request Error Mask
19	ECRCErrMsk	RWS	0		ECRC Error Mask
18	MlfTLPMsk	RWS	0		Malformed TLP Mask
17	RcvOvfMsk	RWS	0		Receiver Overflow Mask
16	UnxCplMsk	RWS	0		Unexpected Completion Mask
15	CplAbrtMsk	RWS	0		Completer Abort Mask
14	CplTOMsk	RWS	0		Completion Timeout Mask
13	FCErrMsk	RWS	0		Flow Control Protocol Error Mask
12	PsnMsk	RWS	0		Poisoned TLP Mask
11:5		R	0		Reserved.
4	DLErrMsk	RWS	0		Data Link Protocol Error Mask
3:1		R	0		Reserved.
0	TrnErrMsk	R	0		Training Error Mask.

13.12.35 Uncorrectable Severity Register

Description

All unreserved bits in these registers are sticky.

Register

R_PcieUncErrSev

Address

0xE_9800_010C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:21		R	0		Reserved
20	URErrSev	RWS	0		Unsupported Request Error Severity
19	ECRCErrSev	RWS	0		ECRC Error Severity
18	MlfTLPSev	RWS	1		Malformed TLP Severity
17	RcvOvfSev	RWS	1		Receiver Overflow Severity
16	UnxCplSev	RWS	0		Unexpected Completion Severity
15	CplAbrtSev	RWS	0		Completer Abort Severity
14	CplTOSev	RWS	0		Completion Timeout Severity
13	FCErrSev	RWS	1		Flow Control Protocol Error Severity
12	PsnSev	RWS	0		Poisoned TLP Severity
11:5		R	0		Reserved.
4	DLErrSev	RWS	1		Data Link Protocol Error Severity
3:1		R	0		Reserved.
0		R	0		Reserved.

13.12.36 Correctable Error Status Register

Description

All unreserved bits in these registers are sticky.

Register

R_PcieCorErrSt

Address

0xE_9800_0110

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:14		R	0		Reserved
13	NFErr	RW1C	0		Advisory NonFatal Error Status
12	RplTOSt	RW1C	0		Replay Timer Timeout Status
11:9		R	0		Reserved.
8	RplRollSt	RW1C	0		Replay_Num Rollover Status
7	BadDLLPSt	RW1C	0		Bad DLLP Status
6	BadTLPSt	RW1C	0		Bad TLP Status
5:1		R	0		Reserved.
0	RcvErrSt	RW1C	0		Receiver Error Status

13.12.37 Correctable Error Mask Register

Description

All unreserved bits in these registers are sticky.

Register

R_PcieCorErrMsk

Attributes

-writeonemixed

Address

0xE_9800_0114

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:14		R	0		Reserved
13	NFErrMask	RW1C	1		Advisory NonFatal Error mask
12	RplTOMsk	RW	0		Replay Timer Timeout Mask
11:9		R	0		Reserved
8	RplRollMsk	RW	0		Replay_Num Rollover Mask
7	BadDLLPMsk	RW	0		Bad DLLP Mask
6	BadTLPMsk	RW	0		Bad TLP Mask
5:1		R	0		Reserved.
0	RcvErrMsk	RW	0		Receiver Error Mask

13.12.38 Advanced Error Capabilities Control Register

Description

Register

R_PcieAdvErrCapCtrl

Address

0xE_9800_0118

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:9		R	0		Reserved
8	ECRCChkEn	RW	0		1 = ECRC Checking on inbound packets enabled. Sticky.
7	ECRCChkCap	R	1		1 = This device is capable of checking ECRC.
6	ECRCGenEn	RW	0		1 = ECRC Generation Enabled. Sticky.
5	ECRCGenCap	R	1		1 = This device is capable of generating ECRC.
4:0	FstErrPtr	R	0		First Error Pointer. Identifies bit position of first error reported in the Uncorrectable Error Status register. Sticky.

13.12.39 Advanced Error Capabilities/Header Log Register (1st Dword)

Description

The Header Log is 4 Dwords and contains the header of the TLP that contained a detected error. All bits in Header Log register(s) are sticky.

Register

R_PcieHdrLog1

Address

0xE_9800_011C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	HdrLog1	R	0		First Header of TLP that contained a detected error. Sticky.

13.12.40 Header Log Register (2nd Dword)

Description

Register

R_PcieHdrLog2

Address

0xE_9800_0120

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	HdrLog2	R	0		Second Header of TLP that contained a detected error. Sticky.

13.12.41 Header Log Register (3rd Dword)

Description

Register

R_PcieHdrLog3

Address

0xE_9800_0124

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	HdrLog3	R	0		Third Header of TLP that contained a detected error. Sticky.

13.12.42 Header Log Register (4th Dword)

Description

Register

R_PcieHdrLog4

Address

0xE_9800_0128

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	HdrLog4	R	0		Fourth Header of TLP that contained a detected error. Sticky.

13.12.43 Root Error Command Register

Description

This register allows the Root Complex to control reporting (ie. generating or disabling interrupts) of incoming ERROR messages.

Register

R_PcieRootErrCmd

Address

0xE_9800_012C

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:3		R	0		Reserved
2	FatErrEn	RW	0		1 = Enable interrupt generation when ERR_FATAL message received. (Errata doc Page 111 says that the Root Port should generate interrupt wire only when Interrupt Disable bit in Command Register is 0 in addition to the above).
1	NFErrEn	RW	0		1 = Enable interrupt generation when ERR_NONFATAL message received. (Errata doc Page 111 says that the Root Port should generate interrupt wire only when Interrupt Disable bit in Command Register is 0 in addition to the above).
0	CorrErrEn	RW	0		1 = Enable interrupt generation when ERR_COR message received. (Errata doc Page 111 says that the Root Port should generate interrupt wire only when Interrupt Disable bit in Command Register is 0 in addition to the above).

13.12.44 Root Error Status Register

Description

The Root Error Status register reports the status of error messages (where these could be ERROR messages received from other devices, or detected by the Root Port itself). Bits 6:0 of this register are sticky.

Register

R_PcieRootErrSt

Address

0xE_9800_0130

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:27	MsgNum	R	0		Since we allocated only 1 MSI interrupt number, this field is irrelevant for us.
26:7		R	0		Reserved.
6	FatErrMsgRcv	RW1C	0		1 = One or more fatal uncorrectable errors detected. This should not happen if SerrEn bit in Command register = 0 (based on Errata doc Page 107).
5	NFErrMsgRcv	RW1C	0		1 = One or more non-fatal uncorrectable errors detected. This should not happen if SerrEn bit in Command register = 0 (based on Errata doc Page 107).
4	FstUncFat	RW1C	0		1 = Bit 2 was set due to a FATAL error.
3	NFErrMul	RW1C	0		1 = Uncorrectable error detected while bit 2 was already set.
2	NFErrRcv	RW1C	0		1 = Uncorrectable error detected (by Root Port or via ERR_(NON)FATAL message). This should not happen if SerrEn bit in Command register = 0 (based on Errata doc Page 107).
1	CorrErrMul	RW1C	0		1 = Correctable error detected while bit 0 was already set.
0	CorErrRcv	RW1C	0		1 = Correctable error detected (by Root Port or via ERR_COR message). This should not happen if SerrEn bit in Command register = 0 (based on Errata doc Page 107).

13.12.45 Root Error Source Identification Register

Description

The Error Source Identification register (all of whose bits are sticky) keeps track of the requester ID of the first such ERROR message for a given category (correctable or uncorrectable).

Register

R_PcieRootErrSrcId

Address

0xE_9800_0134

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:16	UncErrId	R	0		Contains ReqID of uncorrectable error (message) detected when bit 2 is being set
15:0	CorErrId	R	0		Contains ReqID of correctable error (message) detected when bit 0 is being set

13.13 PMI Control and Status Registers

13.13.1 Core Control Register

Register

R_PmiCoreCtrl

Attributes

-noregtest -kernel

Address

0xE_9800_1000

Bit	Mnemonic	Access	Reset	Definition
63:12		R	0	Reserved
11	PERSTN	RWS	0	slot reset
10	PWRRSTN	RWS	0	power reset
9	NSTICKYRSTN	RWS	0	non_sticky reset_n
8	STICKYRSTN	RWS	0	Sticky reset_n
7	CORERSTN	RWS	0	Core reset_n
6	APPREQL1EXIT	RW	0	Application requests L1 exit
5	APPREQL1ENTRY	RW	0	Application requests L1 entry
4		R	0	Reserved
3	APPREQRST	RW	0	Application requests Hot Reset on PCIE link
2	RXLNFLP	RW	0	Rx Lane Flip Enable
1	TXLNFLP	RW	0	Tx Lane Flip Enable
0	ENA	RW	0	Write 1 to start the PCI link training. Typically after reset

13.13.2 PMI Interrupt Summary Register

Description

This register is a summary of the various sources of intterrupts. The source of the interrupt must be cleared to clear bits that are not labelled RW1C. The state of the bits in this register are independent of the R_PmiIntrEn register.

Register

R_PmiIntr

Attributes

-kernel

Address

0xE_9800_1008

Bit	Mnemonic	Access	Reset	Product	Definition
63:34		R	0	ICE9A	Reserved (Overlaps allowed)
63:40		R	0	ICE9B+	Reserved (Overlaps allowed)
39	REQCOMPMULT	RW1C	0	ICE9B+	REQ received multiple errored completions
38	CSIECCMULT	RW1C	0	ICE9B+	CSI detected multiple ECC errors.
37	REQECCMULT	RW1C	0	ICE9B+	REQ detected multiple ECC errors.
36	CCWSYCECCMULT	RW1C	0	ICE9B+	Data from the SYC to the CCW had multiple ECC errors.
35	CCWCSWECCMULT	RW1C	0	ICE9B+	Data from the CSW to the CCW had multiple ECC errors.
34	SYCECCMULT	RW1C	0	ICE9B+	Data from the CSW to the SYC had multiple ECC errors
33	REQRST	R	1		RC requests reset due to link down status.
32		R	0		Reserved
31	DATLINKDWN	R	1		PCI Data Link Layer Down Indication
30	PMTLPBLK	R	0		PM requests blocking of outbound non-completion TLPs.
29	INTD	R	0		INTD Active
28	INTC	R	0		INTC Active
27	INTB	R	0		INTB Active
26	INTA	R	0		INTA Active
25	CORERR	RW1C	0		Received a Correctable Error Message
24	NFERR	RW1C	0		Received a Non-Fatal Error Message
23	FERR	RW1C	0		Received a Fatal Error Message
22	PME	RW1C	0		Received a PM_PME Message
21	TOACK	RW1C	0		Received a PME Turnoff Ack Message
20	VEN	RW1C	0		Received Vendor Message
19	AERINT	R	0		AER INT
18	AERMSI	RW1C	0		AER MSI
17	PMEINT	R	0		PME INT
16	PMEMSI	RW1C	0		PME MSI
15	HPPME	R	0		Hot-plug PME Wake Generation
14	HPINT	R	0		Hot-plug Interrupt
13	HPMSI	RW1C	0		Hot-plug MSI
12		R	0		Reserved
11	CSIADRINT	RW1C	0		CSI detected an out of range address.
10	CSIECCINT	RW1C	0		CSI detected an ECC error.
9		R	0		Reserved
8	CSIWTOINT	RW1C	0		A timeout occurred on the wishbone interface.
7	CSIDBIINT	RW1C	0		A 64 bit access was requested of the DBI.
6		R	0		Reserved
5	REQECCINT	RW1C	0		REQ detected an ECC error.
4	REQCOMPINT	RW1C	0		REQ received an errored completion.
3	CCWSYCECCINT	RW1C	0		Data from the SYC to the CCW had an ECC error.
2	CCWCSWECCINT	RW1C	0		Data from the CSW to the CCW had an ECC error.
1	SYCECCINT	RW1C	0		Data from the CSW to the SYC had an ECC error.
0		R	0		Reserved

13.13.3 PMI Interrupt Enable Register

Register

R_PmiIntrEn

Attributes

-kernel

Address

0xE_9800_1010

Bit	Mnemonic	Access	Reset	Product	Definition
63:34		RW	0	ICE9A	Reserved (Overlaps allowed)
63:40		RW	0	ICE9B+	Reserved (Overlaps allowed)
39	REQCOMPMULT	RW	0	ICE9B+	REQ received multiple errored completions
38	CSIECCMULT	RW	0	ICE9B+	CSI detected multiple ECC errors.
37	REQECCMULT	RW	0	ICE9B+	REQ detected multiple ECC errors.
36	CCWSYCECCMULT	RW	0	ICE9B+	Data from the SYC to the CCW had multiple ECC errors.
35	CCWCSWECCMULT	RW	0	ICE9B+	Data from the CSW to the CCW had multiple ECC errors.
34	SYCECCMULT	RW	0	ICE9B+	Data from the CSW to the SYC had multiple ECC errors
33	REQRST	RW	0		RC requests reset due to link down status.
32	Unused32	RW	0		unused_32
31	DATLINKDWN	RW	0		PCI Data Link Layer Down Indication
30	PMTLPBLK	RW	0		PM requests blocking of outbound non-completion TLPs.
29	INTD	RW	0		INTD Active
28	INTC	RW	0		INTC Active
27	INTB	RW	0		INTB Active
26	INTA	RW	0		INTA Active
25	CORERR	RW	0		Received a Correctable Error Message
24	NFERR	RW	0		Received a Non-Fatal Error Message
23	FERR	RW	0		Received a Fatal Error Message
22	PME	RW	0		Received a PM_PME Message
21	TOACK	RW	0		Received a PME Turnoff Ack Message
20	VEN	RW	0		Received Vendor Message
19	AERINT	RW	0		AER INT
18	AERMSI	RW	0		AER MSI
17	PMEINT	RW	0		PME INT
16	PMEMSI	RW	0		PME MSI
15	HPPME	RW	0		Hot-plug PME Wake Generation
14	HPINT	RW	0		Hot-plug Interrupt
13	HPMSI	RW	0		Hot-plug MSI
12	Unused12	RW	0		unused_12
11	CSIADRINT	RW	0		CSI detected an out of range address.
10	CSIECCINT	RW	0		CSI detected an ECC error.
9	Unused9	RW	0		unused_9
8	CSIWTOINT	RW	0		A timeout occurred on the wishbone interface.
7	CSIDBIINT	RW	0		A 64 bit access was requested of the DBI.
6	Unused6	RW	0		unused_6
5	REQECCINT	RW	0		REQ detected an ECC error.
4	REQCOMPINT	RW	0		REQ received an errored completion.
3	CCWSYCECCINT	RW	0		Data from the SYC to the CCW had an ECC error.
2	CCWCSWECCINT	RW	0		Data from the CSW to the CCW had an ECC error.
1	SYCECCINT	RW	0		Data from the CSW to the SYC had an ECC error.
0	Unused0	RW	0		unused_0

13.13.4 LED Blink Rate Register

Register

R_PmiLedBlinkRate

Attributes

-kernel

Address

0xE_9800_1018

Bit	Mnemonic	Access	Reset	Definition
63:32				Reserved
31:0	BLINKRATE	RW	0	Count in ICLK of the PWWR and ATTN indicators. (This count defines the high (and the low) time of the 50% duty cycle blink rate.

13.13.5 Send Unlock Message Register

Register

R_PmiSndUnlkMsg

Address

0xE_9800_1028

Bit	Mnemonic	Access	Reset	Definition
63:1				Reserved
0	GO	W1C	0	Write 1 to send an unlock message out. Self-clearing bit

13.13.6 Send Turnoff Message Register

Register

R_PmiSndTrnOffMsg

Address

0xE_9800_1030

Bit	Mnemonic	Access	Reset	Definition
63:1				Reserved
0	GO	W1C	0	Write 1 to send a turn-off message out. Self-clearing bit

13.13.7 Link Status Register

Register

R_PmiLinkStat

Attributes

-kernel

Address

0xE_9800_1038

Bit	Mnemonic	Access	Reset	Definition
63:15				Reserved
14:12	PMDS	R	0x4	Power Management D-State
11:7	LTSSMCS	R	0	Link Training and Status State Machine Current State
6:4	PMCS	R	0	Power Management Current State
3	DATLK	R	0	PCI Data Link Layer Up/Down Indication
2	REQRST	R	1	RC requests reset due to link down status.
1	PHYLK	R	0	PCI Phy Link Up/Down Indication
0	PMTLPBLK	R	0	Power management control to block schedule of new TLP requests

13.13.8 Root-Complex Debug Info

Register

R_PmiRcDbg

Address

0xE_9800_1040

Bit	Mnemonic	Access	Reset	Definition
63:12		R	0	Reserved
11	XSCRDIS	R	0	Transmit Scrambler Disabled
10	XLKDIS	R	1	Transmit Link Disabled
9	XLKTRN	R	0	Transmit Link In Training
8:3				Reserved
2	DETLOOP	R	0	PIPE TxDetextRx/Loopback on. PHY is doing a receiver detection or is in loopback mode
1	TXEIDLE	R	0	PIPE TxElecIdle on. PHYtransmits electrical idle
0	TXCOMPL	R	0	PIPE TxCompliance on. PHY transmits compliance patterns

13.13.9 Force Ecc Error Register

Description

Used to artificially cause single bit ECC errors at various generators within the PMI.

Register

R_PmiFrcEccErr

Address

0xE_9800_2000

Bit	Mnemonic	Access	Reset	Definition
63:5				Reserved
4	EnECCCorr	RW	1	Enable correction of ECC errors.
3	SycBadDat1	RWS	0	Flip bit 1 of word 0 of data coming out of the SYC write buffer.
2	SycBadDat0	RWS	0	Flip bit 0 of word 0 of data coming out of the SYC write buffer.
1	CswBadDat1	RWS	0	Flip bit 1 of word 0 of data going out on the CSW.
0	CswBadDat0	RWS	0	Flip bit 0 of word 0 of data going out on the CSW.

13.13.10 CSI Ecc Error Register

Description

Debug information in the event an ECC error was detected by the CSI. This is for data coming from the CSW to the CSI.

Register

R_PmiCsiEccErr

Address

0xE_9800_2018

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:54		R	0	Reserved
53	Dbe	R	0	It was a double bit error.
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding multi-interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36	Synd	R	0	Syndrome of the errored data.
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.11 CSI Address Error Register

Description

Debug information in the event an out of range address was detected by the CSI. This is for commands coming from the CSW that are in the gross CSI range, but not in the range of any specific function within the CSI. To wit, the following must be true for the address to be valid:
Addr[35:12] = 0xE980xx where xx <= 0x30
or
the address is within the IoSCB space
or
the address is within the I2C space
or
the address is within the Uart space

Register

R_PmiCsiAdrErr

Address

0xE_9800_2020

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:53		R	0	Reserved
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36		R	0	Reserved
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.12 DBI 64bit Access Error Register

Description

Debug information in the event a 64 bit access to the DBI was detected by the CSI.

Register

R_PmiCsiDbiErr

Address

0xE_9800_2028

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:53		R	0	Reserved
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36		R	0	Reserved
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.13 CSI Wishbone Timeout Error Register

Description

Debug information in the event a timeout occurred in a Wishbone transaction.

Register

R_PmiCsiWtoErr

Address

0xE_9800_2030

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:53		R	0	Reserved
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36		R	0	Reserved
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.14 REQ Ecc Error Register

Description

Debug information in the event an ECC error was detected by the REQ. This is for data coming from the CSW to the REQ.

Register

R_PmiReqEccErr

Address

0xE_9800_2040

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:54		R	0	Reserved
53	Dbe	R	0	It was a double bit error
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding multi-interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36	Synd	R	0	Syndrome of the errored data.
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.15 REQ Completion Error Register

Description

Debug information in the event an errored completion was received by the REQ from the Root Complex.

Register

R_PmiReqCompErr

Attributes

-kernel

Address

0xE_9800_2048

Bit	Mnemonic	Access	Reset	Definition
63:60	Reas	R	0	Reason code for errored completion.
59:53		R	0	Reserved
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding multi-interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36		R	0	Reserved
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.16 SYC CSW Ecc Error Register

Description

Debug information in the event an ECC error was detected by the SYC. This is for data coming from the CSW to the SYC. The address given is the PCI address of the completion or of the current completion segment.

Register

R_PmiSycCswEccErr

Address

0xE_9800_2050

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:54		R	0	Reserved
53	Dbe	R	0	It was a double bit error
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding multi-interrupt bit in the summary register is cleared.
51:44		R	0	Reserved
43:36	Synd	R	0	Syndrome of the errored data.
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.17 CCW CSW Ecc Error Register

Description

Debug information in the event an ECC error was detected by the CCW. This is for data coming from the CSW to the CCW.

Register

R_PmiCcwCswEccErr

Address

0xE_9800_2060

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:54		R	0	Reserved
53	Dbe	R	0	It was a double bit error
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding multi-interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36	Synd	R	0	Syndrome of the errored data.
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.18 CCW SYC Ecc Error Register

Description

Debug information in the event an ECC error was detected by the CCW. This is for data coming from the SYC to the CCW.

Register

R_PmiCcwSycEccErr

Address

0xE_9800_2068

Bit	Mnemonic	Access	Reset	Definition
63:60		R	0	Reserved
59:54		R	0	Reserved
53	Dbe	R	0	It was a double bit error
52	Mult	R	0	Multiple Errors received since last serviced. Cleared when the corresponding multi-interrupt bit in the summary register is cleared.
51:44	Origin	R	0	The origin of the errored transaction.
43:36	Synd	R	0	Syndrome of the errored data.
35:3	Addr	R	0	Address of the errored transaction.
2:0		R	0	Reserved

13.13.19 MSI Address Register

Register

R_PmiMsiAddr

Attributes

-kernel

Address

0xE_9800_3000

Bit	Mnemonic	Access	Reset	Definition
63:6	Addr	RW	0	Base address for the MSI range.
5:0		R	0	Reserved

13.13.20 Wishbone Timeout Value Register

Register

R_PmiWbToVal

Address

0xE_9800_3008

Bit	Mnemonic	Access	Reset	Definition
63:8		R	0	Reserved
7:0	WBTOVAL	RW	0x10	Timeout value in CCLKs

13.13.21 VSM Request Double Word 1 and 2 Register

Register

R_PmiVmReqDW12

Address

0xE_9801_0000

Bit	Mnemonic	Access	Reset	Definition
63:56	Unused1	RW	0	Unused 1
55:48	CODE	RW	0	Code field
47:40	Unused0	RW	0	Unused 0
39:24	REQID	RW	0	Requestor id
23	TD	RW	0	Digest present
22	EP	RW	0	Poisoned indicator
21:20	ATTR	RW	0	Attribute Field
19:10	LEN	RW	0	Length Field. Valid values are 0 and 1. Other values will default to 1.
9:7	TC	RW	0	Traffic Class
6:5	FMT	RW	0	Format field
4:0	TYPE	RW	0	Type field

13.13.22 VSM Request Double Word 3 and 4 Register

Register

R_PmiVmReqDW34

Address

0xE_9801_0008

Bit	Mnemonic	Access	Reset	Definition
63:0	ADDR	RW	0	64 bit address

13.13.23 VMI Request Data Register

Register

R_PmiVmReqDat

Attributes

-writeonemixed

Address

0xE_9801_0010

Bit	Mnemonic	Access	Reset	Definition
63:33		R	0	Reserved
32	GO	W1C	0	Write 1 to initiate a Vendor Message Request. Software needs to setup the Vendor Message Data Registers and the Vendor Message Header Register before setting this flag. This is a self-resetting flag
31:0	DAT	RW	0	Optional data for the request

13.13.24 Received Vendor Message Double Word 1 and 2 Register

Register

R_PmiRcvVenMsgDW12

Address

0xE_9801_0018

Bit	Mnemonic	Access	Reset	Definition
63:56		R	0	Reserved
55:48	CODE	R	0	Code field
47:40	TAG	R	0	Tag field
39:24	REQID	R	0	Requestor id
23	TD	R	0	Digest present. PRC has been configured to strip ECRC, hence this bit will likely always be 0.
22	EP	R	0	Poisoned indicator
21:20	ATTR	R	0	Attribute Field
19:10	LEN	R	0	Length Field
9:7	TC	R	0	Traffic Class
6:5	FMT	R	0	Format field
4:0	TYPE	R	0	Type field

13.13.25 Received Vendor Message Double Word 3 and 4 Register

Register

R_PmiRcvVenMsgDW34

Address

0xE_9801_0020

Bit	Mnemonic	Access	Reset	Definition
63:0	ADDR	R	0	64 bit address

13.13.26 Received Vendor Message Payload Register

Register

R_PmiRcvVenMsgPld

Address

0xE_9801_0028

Bit	Mnemonic	Access	Reset	Definition
63:34		R	0	Reserved
33	OVFLW	R	0	Received Vendor Message Overflow. Set if a Vendor Message received before previous message was serviced
31:0	PAYLD	R	0	Received Vendor Message Payload

13.14 PCI Express Phy Registers

All of the registers in this section are within the PCI Express Phy. The contents of these registers come from the PCIe1 90nM PHY Core Data Book.

13.14.1 Less Than Limit Compare Point Register

Description

Less Than Limit Compare point

Register

R_PciePhyCrClockCrcmpLtLimit

Address

0xE98100008

Bit	**Mnemonic**	**Access**	**Reset**	**Type**	**Definition**
15:0	CrcmpLtLimit	RW	0x0		Less Than Limit Compare point.

13.14.2 Greater Than Limit Compare Point Register

Description

Greater Than Limit Compare point

Register

R_PciePhyCrClockCrcmpGtLimit

Address

0xE98100010

Bit	**Mnemonic**	**Access**	**Reset**	**Type**	**Definition**
15:0	CrcmpGtLimit	RW	0xFFFF		Greater Than Limit Compare point.

13.14.3 Compare/Scratch Value Mask Register

Description

Compare/Scratch value mask

Register

R_PciePhyCrClockCrcmpMask

Address

0xE98100018

Bit	**Mnemonic**	**Access**	**Reset**	**Type**	**Definition**
15:0	CrcmpMask	RW	0xFFFF		Compare/Scratch value mask.

13.14.4 Scratch Space Control Register

Description

Scratch space control bits

Register

R_PciePhyCrClockCrcmpCtl

Address

0xE98100020

Bit	Mnemonic	Access	Reset	Type	Definition
1	HoldScratch1	RW	0		Don t update Scratch1 on register reads.
0	HoldScratch0	RW	0		Don t update Scratch0 on register reads.

13.14.5 Scratch Register Comparisons To Limits Results Register

Description

Results of scratch register comparisons to limits

Register

R_PciePhyCrClockCrcmpStat

Address

0xE98100028

Bit	Mnemonic	Access	Reset	Type	Definition
5	S1S0Outside	RS	X		Logical OR of S1_S0_LOW and S1_S0_HIGH useful to determine if the difference is.
4	S0Outside	RS	X		Logical OR of S0_LOW and S0_HIGH useful to determine if the value is near signed.
3	S1S0High	RS	X		Masked(Scratch1-Scratch0) is higher than CRCMP_HT_LIMIT.
2	S1S0Low	RS	X		Masked(Scratch1-Scratch0) is lower than CRCMP_LT_LIMIT.
1	S0High	RS	X		Masked Scratch0 is higher than CRCMP_HT_LIMIT.
0	S0Low	RS	X		Masked Scratch0 is lower than CRCMP_LT_LIMIT.

13.14.6 Number Of Samples To Count Register

Description

Number of samples to count

Register

R_PciePhyCrClockScopeSamples

Address

0xE98100030

Bit	Mnemonic	Access	Reset	Type	Definition
15:0	ScopeSamples	RW	0x100		Number of samples to count.

13.14.7 Scope Counting Results Register

Description

Results of scope counting A write to this register will start the counting process The value of FFFF indicates counting still in progress

Register

R_PciePhyCrClockScopeCount

Address

0xE98100038

Bit	Mnemonic	Access	Reset	Type	Definition
15:0	ScopeCount	RS	X		Results of scope counting A write to this register will start the counting proce.

13.14.8 Support DAC Values And Controls Register

Description

Support DAC values and controls

Register

R_PciePhyCrClockDacCtl

Address

0xE98100040

Bit	Mnemonic	Access	Reset	Type	Definition
14:12	DacMode	RW	0x0		DAC output mode 0 - powered down 1 - unused 2 - hi-range margining (VP25*418e-6.
11	OvrdRtuneRx	RW	0		Write DAC_VAL[5:0] to the Rx rtune bus.
10	OvrdRtuneTx	RW	0		Write DAC_VAL[5:0] to the Tx rtune bus.
9:0	DacVal	RW	0x1FF		Digital value to use for DAC.

13.14.9 Resistor Tuning Controls Register

Description

Resistor tuning controls

Register

R_PciePhyCrClockRtuneCtl

Address

0xE98100048

Bit	Mnemonic	Access	Reset	Type	Definition
10	AdcTrig	RW	0		Trigger ADC conversion.
9	RtuneTrig	RW	0		Trigger manual resistor calibration.
8	RtuneDis	RW	0		Disable automatic resistor recalibrations.
7	CmpInvert	RW	0		Invert output of comparator (to reverse SAR feed- back loop).
6	DacChop	RW	0		Polarity of chop control for DAC.
5	RscX4	RW	1		Set x4 in rescal circuitry.
4	SelAtbp	RW	0		Select atb_s_p for A/D measurement.
3	PwronLcl	RW	0		Value of poweron to force.
2	FrcPwron	RW	0		Override internal poweron.
1:0	Mode	RW	0x0		Restune SAR mode 0 - normal restune 1 - ADC 2 - Rx Resistor test 3 - Tx Resistor.

13.14.10 ADC Process Results Register

Description

Results of ADC process A read from this register starts a new A/D conversion

Register

R_PciePhyCrClockAdcOut

Address

0xE98100050

Bit	Mnemonic	Access	Reset	Type	Definition
10	Fresh	RS	X		Flag indicates that a new A/D conversion result is present.
9:0	Value	RS	X		A/D conversion result Based on RTUNE_CTL.

13.14.11 Current MPLL Phase Selector Value Register

Description

Current MPLL phase selector value

Register

R_PciePhyCrClockSsPhase

Address

0xE98100058

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
12	ZeroFreq	RWS	0		Current MPLL phase selector value Must be set for PHASE writes to stick.
11:2	Val	RWS	0x0		Current MPLL phase selector value.
1:0	Dthr	RWS	0x0		Current MPLL phase selector value.

13.14.12 JTAG Chip ID Register (Lower 16 Bits)

Description

Internal Chip ID used by JTAG - upper 16 bits Not unique between UP3_1.0 parts

Register

R_PciePhyCrClockChipIdHi

Address

0xE98100060

Bit	Mnemonic	Access	Reset	Type	Definition
15:0	ChipIdHi	R	0x3005		Internal Chip ID used by JTAG - upper 16 bits Not unique between UP3_1.

13.14.13 JTAG Chip ID Register (Upper 16 Bits)

Description

Internal Chip ID used by JTAG - lower 16 bits Not unique between UP3_1.0 parts

Register

R_PciePhyCrClockChipIdLo

Address

0xE98100068

Bit	Mnemonic	Access	Reset	Type	Definition
15:0	ChipIdLo	R	0x4CD		Internal Chip ID used by JTAG - lower 16 bits Not unique between UP3_1.

13.14.14 Frequency Control Inputs Status Register

Description

Status of Frequency control inputs Reset value depends on inputs

Register

R_PciePhyCrClockFreqStat

Address

0xE98100070

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved	RS	X		Always reads as 1.
14:13	Prescale	RS	X		Prescaler control.
12:8	Ncy	RS	X		Divide by 4 cycle control.
7:6	Ncy5	RS	X		Divide by 5 control.
5:3	IntCtl	RS	X		Integral charge pump control.
2:0	PropCtl	RS	X		Proportional charge pump control.

13.14.15 Various Control Inputs Status Register

Description

Status of various control inputs Reset value depends on inputs

Register

R_PciePhyCrClockCtlStat

Address

0xE98100078

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14	FastTech	RS	X		Technology is fast.
13	VpIs1p2	RS	X		Low voltage supply is 1.
12	VphIs3p3	RS	X		High voltage supply is 3.
11	WideXface	RS	X		Wide interface control.
10	RtuneDoTune	RS	X		Manual resistor tune control.
9	Reserved	RS	X		Always reads as 1.
8:6	CkoWordCon	RS	X		Cko_word mux control.
5:4	CkoAliveCon	RS	X		Cko_alive mux control.
3	MpllSsEn	RS	X		Spread spectrum enable.
2	MpllPwron	RS	X		Mpll power-on control.
1	MpllClkOff	RS	X		Reference clock is off.
0	UseRefclkAlt	RS	X		Use alternate refclk.

13.14.16 Level Control Inputs Status Register

Description

Status of level control inputs Reset value depends on inputs

Register

R_PciePhyCrClockLvlStat

Address

0xE98100080

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved	RS	X		Always reads as 1.
14:10	TxLvl	RS	X		Transmit level.
9:5	LosLvl	RS	X		Loss of Signal Detector level.
4:0	AcjtLvl	RS	X		AC JTag Comparator level.

13.14.17 Creg Control I/O Status Register

Description

Status of creg control I/O Reset value depends on inputs

Register

R_PciePhyCrClockCregStat

Address

0xE98100088

Bit	Mnemonic	Access	Reset	Type	Definition
8	Reserved1	RS	X		Always reads as 1.
7	OpDone	RS	X		Operation is complete output.
6	PowerGood	RS	X		Power good output.
5	CrAck	RS	X		Creg request Acknowledgement.
4	Reserved	RS	X		Always reads as 1.
3	CrCapAddr	RS	X		Capture Address request.
2	CrCapData	RS	X		Capture Data request.
1	CrWrite	RS	X		Write request.
0	CrRead	RS	X		Read request.

13.14.18 Frequency Control Inputs Override Register

Description

Override of Frequency control inputs

Register

R_PciePhyCrClockFreqOvrd

Address

0xE98100090

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	Prescale	RW	0x2		Prescaler control 00 - no scaling 01 - double refclk freq 10 - halve refclk freq.
12:8	Ncy	RW	0x5		Divide by 4 cycle control MPLL Divider Period=4*(NCY+1)+NCY5 Valid only when NCY.
7:6	Ncy5	RW	0x1		Divide by 5 control MPLL Divider Period=4*(NCY+1)+NCY5 Valid only when NCY5<=NCY.
5:3	IntCtl	RW	0x0		Integral charge pump control Integral current = (n+1)/8*full_scale.
2:0	PropCtl	RW	0x7		Proportional charge pump control Proportional current = (n+1)/8*full_scale.

13.14.19 Various Control Inputs Override Register

Description

Override of various control inputs

Register

R_PciePhyCrClockCtlOvrd

Address

0xE98100098

Bit	Mnemonic	Access	Reset	Type	Definition
15	OvrdStatic	RWS	0		Override static controls (bits 14:10).
14	FastTech	RW	0		Technology is fast.
13	VpIs1p2	RW	0		Low voltage supply is 1.
12	VphIs3p3	RW	0		High voltage supply is 3.
11	WideXface	RW	1		Wide interface control.
10	RtuneDoTune	RW	0		Manual resistor tune control.
9	OvrdClk	RWS	0		Override clock controls (bits 8:0).
8:6	CkoWordCon	RW	0x1		Cko_word mux control.
5:4	CkoAliveCon	RW	0x1		Cko_alive mux control.
3	MpllSsEn	RW	0		Spread spectrum enable.
2	MpllPwron	RW	1		Mpll power-on control.
1	MpllClkOff	RW	0		Reference clock is off.
0	UseRefclkAlt	RW	0		Use alternate refclk.

13.14.20 Level Control Inputs Override Register

Description

Override of level control inputs

Register

R_PciePhyCrClockLvlOvrd

Address

0xE981000A0

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Override all level controls.
14:10	TxLvl	RW	0x10		Transmit level.
9:5	LosLvl	RW	0x10		Loss of Signal Detector level.
4:0	AcjtLvl	RW	0x10		AC JTag Comparator level.

13.14.21 Creg Control I/O Override Register

Description

Override of creg control I/O

Register

R_PciePhyCrClockCregOvrd

Address

0xE981000A8

Bit	Mnemonic	Access	Reset	Type	Definition
8	OvrdOut	RWS	0		Override outputs (bits 7:5).
7	OpDone	RW	0		Operation is complete output.
6	PowerGood	RW	1		Power good output.
5	CrAck	RW	0		Creg request Acknowledgement.
4	OvrdIn	RWS	0		Override inputs (bits 3:0).
3	CrCapAddr	RW	0		Capture Address request.
2	CrCapData	RW	0		Capture Data request.
1	CrWrite	RW	0		Write request.
0	CrRead	RW	0		Read request.

13.14.22 MPLL Controls Register

Description

MPLL Controls

Register

R_PciePhyCrClockMpllCtl

Address

0xE981000B0

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - mpll_gear_shift 2 - mpll.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - mpll_gear_shift 2 - mpll.
4	RefclkDelay	RW	0		Delay refclk output of prescaler.
3	DisParaCreg	RW	0		Disable Parallel creg xface.
2	OvrdClkdrv	RWS	0		Override clock driver controls.
1	ClkdrvDig	RW	0		Value for digital clock drivers.
0	ClkdrvAna	RW	0		Value for analog clock drivers.

13.14.23 MPLL Test Controls Register

Description

MPLL Test Controls

Register

R_PciePhyCrClockMpllTst

Address

0xE981000B8

Bit	Mnemonic	Access	Reset	Type	Definition
15	OvrdCtl	RWS	0		Override MPLL reset and gearshift controls.
14	GearshiftVal	RW	0		Value to override for mpll_gearshift.
13	ResetVal	RW	0		Value to override for mpll_reset.
12:2	MeasIv	RW	0x0		Measure various mpll controls bit 12 - enable phase linearity testing of phase i.
1	MeasGd	RW	0		Measure GD Should be set when various MEAS_IV bits are set for correct measureme.
0	AtbSense	RW	0		Hook up ATB sense lines.

13.14.24 Transmit Control Inputs Status Register (Lane 0)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane0TxStat

Address

0xE98110008

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.25 Receiver Control Inputs Status Register (Lane 0)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane0RxStat

Address

0xE98110010

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.26 Output Signals Status Register (Lane 0)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane0OutStat

Address

0xE98110018

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.27 Transmitter Control Inputs Override Register (Lane 0)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane0TxOvrd

Address

0xE98110020

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.28 Receiver Control Inputs Override Register (Lane 0)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane0RxOvrd

Address

0xE98110028

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.29 Output Signals Override Register (Lane 0)

Description

Override of output signals

Register

R_PciePhyCrLane0OutOvrd

Address

0xE98110030

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.30 Debug Control Register (Lane 0)

Description

Debug control register

Register

R_PciePhyCrLane0DbgCtl

Address

0xE98110038

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.31 Pattern Generator Controls Register (Lane 0)

Description

Pattern Generator controls

Register

R_PciePhyCrLane0PgCtl

Address

0xE98110080

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.32 Pattern Matcher Controls Register (Lane 0)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane0PmCtl

Address

0xE981100C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.33 Pattern Match Error Counter Register (Lane 0)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane0PmErr

Address

0xE981100C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.34 Current Phase Selector Value. Register (Lane 0)

Description

Current phase selector value.

Register

R_PciePhyCrLane0Phase

Address

0xE981100D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.35 Current Frequency Integrator Value. Register (Lane 0)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane0Freq

Address

0xE981100D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.36 Scope Control Register (Lane 0)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane0ScopeCtl

Address

0xE981100E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.37 Recovered Domain Receiver Control Register (Lane 0)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane0RxCtl

Address

0xE981100E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.38 Receiver Debug Register (Lane 0)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane0RxDbg

Address

0xE981100F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.39 RX Control Register (Lane 0)

Description

RX Control Bits

Register

R_PciePhyCrLane0RxAnaCtrl

Address

0xE98110180

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.40 RX ATB Register (Lane 0)

Description

RX ATB bits

Register

R_PciePhyCrLane0RxAnaAtb

Address

0xE98110188

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.41 8 Bit Programming Register (Lane 0)

Description

8 bit programming register

Register

R_PciePhyCrLane0PllPrg2

Address

0xE98110190

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.42 10 Bit Programming Register (Lane 0)

Description

10 bit programming register

Register

R_PciePhyCrLane0PllPrg1

Address

0xE98110198

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.43 10 Bit Programming Register (Lane 0)

Description

10 bit programming register

Register

R_PciePhyCrLane0PllMeas

Address

0xE981101A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.44 TX ATB Control Register (Set 1) (Lane 0)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane0TxAnaAtbsel1

Address

0xE981101A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.45 TX ATB Control Register (Set 2) (Lane 0)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane0TxAnaAtbsel2

Address

0xE981101B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.46 TX POWER STATE Control Register (Lane 0)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane0TxAnaControl

Address

0xE981101B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.47 Transmit Control Inputs Status Register (Lane 1)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane1TxStat

Address

0xE98110808

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.48 Receiver Control Inputs Status Register (Lane 1)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane1RxStat

Address

0xE98110810

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.49 Output Signals Status Register (Lane 1)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane1OutStat

Address

0xE98110818

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.50 Transmitter Control Inputs Override Register (Lane 1)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane1TxOvrd

Address

0xE98110820

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.51 Receiver Control Inputs Override Register (Lane 1)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane1RxOvrd

Address

0xE98110828

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.52 Output Signals Override Register (Lane 1)

Description

Override of output signals

Register

R_PciePhyCrLane1OutOvrd

Address

0xE98110830

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.53 Debug Control Register (Lane 1)

Description

Debug control register

Register

R_PciePhyCrLane1DbgCtl

Address

0xE98110838

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.54 Pattern Generator Controls Register (Lane 1)

Description

Pattern Generator controls

Register

R_PciePhyCrLane1PgCtl

Address

0xE98110880

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.55 Pattern Matcher Controls Register (Lane 1)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane1PmCtl

Address

0xE981108C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.56 Pattern Match Error Counter Register (Lane 1)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane1PmErr

Address

0xE981108C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.57 Current Phase Selector Value. Register (Lane 1)

Description

Current phase selector value.

Register

R_PciePhyCrLane1Phase

Address

0xE981108D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.58 Current Frequency Integrator Value. Register (Lane 1)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane1Freq

Address

0xE981108D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.59 Scope Control Register (Lane 1)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane1ScopeCtl

Address

0xE981108E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.60 Recovered Domain Receiver Control Register (Lane 1)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane1RxCtl

Address

0xE981108E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.61 Receiver Debug Register (Lane 1)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane1RxDbg

Address

0xE981108F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.62 RX Control Register (Lane 1)

Description

RX Control Bits

Register

R_PciePhyCrLane1RxAnaCtrl

Address

0xE98110980

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.63 RX ATB Register (Lane 1)

Description

RX ATB bits

Register

R_PciePhyCrLane1RxAnaAtb

Address

0xE98110988

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.64 8 Bit Programming Register (Lane 1)

Description

8 bit programming register

Register

R_PciePhyCrLane1PllPrg2

Address

0xE98110990

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.65 10 Bit Programming Register (Lane 1)

Description

10 bit programming register

Register

R_PciePhyCrLane1PllPrg1

Address

0xE98110998

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.66 10 Bit Programming Register (Lane 1)

Description

10 bit programming register

Register

R_PciePhyCrLane1PllMeas

Address

0xE981109A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.67 TX ATB Control Register (Set 1) (Lane 1)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane1TxAnaAtbsel1

Address

0xE981109A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.68 TX ATB Control Register (Set 2) (Lane 1)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane1TxAnaAtbsel2

Address

0xE981109B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.69 TX POWER STATE Control Register (Lane 1)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane1TxAnaControl

Address

0xE981109B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.70 Transmit Control Inputs Status Register (Lane 2)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane2TxStat

Address

0xE98111008

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.71 Receiver Control Inputs Status Register (Lane 2)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane2RxStat

Address

0xE98111010

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.72 Output Signals Status Register (Lane 2)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane2OutStat

Address

0xE98111018

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.73 Transmitter Control Inputs Override Register (Lane 2)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane2TxOvrd

Address

0xE98111020

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.74 Receiver Control Inputs Override Register (Lane 2)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane2RxOvrd

Address

0xE98111028

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.75 Output Signals Override Register (Lane 2)

Description

Override of output signals

Register

R_PciePhyCrLane2OutOvrd

Address

0xE98111030

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.76 Debug Control Register (Lane 2)

Description

Debug control register

Register

R_PciePhyCrLane2DbgCtl

Address

0xE98111038

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.77 Pattern Generator Controls Register (Lane 2)

Description

Pattern Generator controls

Register

R_PciePhyCrLane2PgCtl

Address

0xE98111080

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.78 Pattern Matcher Controls Register (Lane 2)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane2PmCtl

Address

0xE981110C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.79 Pattern Match Error Counter Register (Lane 2)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane2PmErr

Address

0xE981110C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.80 Current Phase Selector Value. Register (Lane 2)

Description

Current phase selector value.

Register

R_PciePhyCrLane2Phase

Address

0xE981110D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.81 Current Frequency Integrator Value. Register (Lane 2)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane2Freq

Address

0xE981110D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.82 Scope Control Register (Lane 2)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane2ScopeCtl

Address

0xE981110E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.83 Recovered Domain Receiver Control Register (Lane 2)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane2RxCtl

Address

0xE981110E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.84 Receiver Debug Register (Lane 2)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane2RxDbg

Address

0xE981110F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.85 RX Control Register (Lane 2)

Description

RX Control Bits

Register

R_PciePhyCrLane2RxAnaCtrl

Address

0xE98111180

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.86 RX ATB Register (Lane 2)

Description

RX ATB bits

Register

R_PciePhyCrLane2RxAnaAtb

Address

0xE98111188

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.87 8 Bit Programming Register (Lane 2)

Description

8 bit programming register

Register

R_PciePhyCrLane2PllPrg2

Address

0xE98111190

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.88 10 Bit Programming Register (Lane 2)

Description

10 bit programming register

Register

R_PciePhyCrLane2PllPrg1

Address

0xE98111198

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.89 10 Bit Programming Register (Lane 2)

Description

10 bit programming register

Register

R_PciePhyCrLane2PllMeas

Address

0xE981111A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.90 TX ATB Control Register (Set 1) (Lane 2)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane2TxAnaAtbsel1

Address

0xE981111A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.91 TX ATB Control Register (Set 2) (Lane 2)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane2TxAnaAtbsel2

Address

0xE981111B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.92 TX POWER STATE Control Register (Lane 2)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane2TxAnaControl

Address

0xE981111B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.93 Transmit Control Inputs Status Register (Lane 3)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane3TxStat

Address

0xE98111808

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.94 Receiver Control Inputs Status Register (Lane 3)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane3RxStat

Address

0xE98111810

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.95 Output Signals Status Register (Lane 3)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane3OutStat

Address

0xE98111818

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.96 Transmitter Control Inputs Override Register (Lane 3)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane3TxOvrd

Address

0xE98111820

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.97 Receiver Control Inputs Override Register (Lane 3)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane3RxOvrd

Address

0xE98111828

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.98 Output Signals Override Register (Lane 3)

Description

Override of output signals

Register

R_PciePhyCrLane3OutOvrd

Address

0xE98111830

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.99 Debug Control Register (Lane 3)

Description

Debug control register

Register

R_PciePhyCrLane3DbgCtl

Address

0xE98111838

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.100 Pattern Generator Controls Register (Lane 3)

Description

Pattern Generator controls

Register

R_PciePhyCrLane3PgCtl

Address

0xE98111880

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.101 Pattern Matcher Controls Register (Lane 3)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane3PmCtl

Address

0xE981118C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.102 Pattern Match Error Counter Register (Lane 3)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane3PmErr

Address

0xE981118C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.103 Current Phase Selector Value. Register (Lane 3)

Description

Current phase selector value.

Register

R_PciePhyCrLane3Phase

Address

0xE981118D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.104 Current Frequency Integrator Value. Register (Lane 3)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane3Freq

Address

0xE981118D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.105 Scope Control Register (Lane 3)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane3ScopeCtl

Address

0xE981118E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.106 Recovered Domain Receiver Control Register (Lane 3)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane3RxCtl

Address

0xE981118E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.107 Receiver Debug Register (Lane 3)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane3RxDbg

Address

0xE981118F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.108 RX Control Register (Lane 3)

Description

RX Control Bits

Register

R_PciePhyCrLane3RxAnaCtrl

Address

0xE98111980

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.109 RX ATB Register (Lane 3)

Description

RX ATB bits

Register

R_PciePhyCrLane3RxAnaAtb

Address

0xE98111988

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.110 8 Bit Programming Register (Lane 3)

Description

8 bit programming register

Register

R_PciePhyCrLane3PllPrg2

Address

0xE98111990

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.111 10 Bit Programming Register (Lane 3)

Description

10 bit programming register

Register

R_PciePhyCrLane3PllPrg1

Address

0xE98111998

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.112 10 Bit Programming Register (Lane 3)

Description

10 bit programming register

Register

R_PciePhyCrLane3PllMeas

Address

0xE981119A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.113 TX ATB Control Register (Set 1) (Lane 3)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane3TxAnaAtbsel1

Address

0xE981119A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.114 TX ATB Control Register (Set 2) (Lane 3)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane3TxAnaAtbsel2

Address

0xE981119B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.115 TX POWER STATE Control Register (Lane 3)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane3TxAnaControl

Address

0xE981119B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.116 Transmit Control Inputs Status Register (Lane 4)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane4TxStat

Address

0xE98112008

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.117 Receiver Control Inputs Status Register (Lane 4)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane4RxStat

Address

0xE98112010

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.118 Output Signals Status Register (Lane 4)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane4OutStat

Address

0xE98112018

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.119 Transmitter Control Inputs Override Register (Lane 4)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane4TxOvrd

Address

0xE98112020

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.120 Receiver Control Inputs Override Register (Lane 4)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane4RxOvrd

Address

0xE98112028

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.121 Output Signals Override Register (Lane 4)

Description

Override of output signals

Register

R_PciePhyCrLane4OutOvrd

Address

0xE98112030

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.122 Debug Control Register (Lane 4)

Description

Debug control register

Register

R_PciePhyCrLane4DbgCtl

Address

0xE98112038

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.123 Pattern Generator Controls Register (Lane 4)

Description

Pattern Generator controls

Register

R_PciePhyCrLane4PgCtl

Address

0xE98112080

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.124 Pattern Matcher Controls Register (Lane 4)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane4PmCtl

Address

0xE981120C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.125 Pattern Match Error Counter Register (Lane 4)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane4PmErr

Address

0xE981120C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.126 Current Phase Selector Value. Register (Lane 4)

Description

Current phase selector value.

Register

R_PciePhyCrLane4Phase

Address

0xE981120D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.127 Current Frequency Integrator Value. Register (Lane 4)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane4Freq

Address

0xE981120D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.128 Scope Control Register (Lane 4)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane4ScopeCtl

Address

0xE981120E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.129 Recovered Domain Receiver Control Register (Lane 4)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane4RxCtl

Address

0xE981120E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.130 Receiver Debug Register (Lane 4)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane4RxDbg

Address

0xE981120F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.131 RX Control Register (Lane 4)

Description

RX Control Bits

Register

R_PciePhyCrLane4RxAnaCtrl

Address

0xE98112180

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.132 RX ATB Register (Lane 4)

Description

RX ATB bits

Register

R_PciePhyCrLane4RxAnaAtb

Address

0xE98112188

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.133 8 Bit Programming Register (Lane 4)

Description

8 bit programming register

Register

R_PciePhyCrLane4PllPrg2

Address

0xE98112190

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.134 10 Bit Programming Register (Lane 4)

Description

10 bit programming register

Register

R_PciePhyCrLane4PllPrg1

Address

0xE98112198

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.135 10 Bit Programming Register (Lane 4)

Description

10 bit programming register

Register

R_PciePhyCrLane4PllMeas

Address

0xE981121A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.136 TX ATB Control Register (Set 1) (Lane 4)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane4TxAnaAtbsel1

Address

0xE981121A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.137 TX ATB Control Register (Set 2) (Lane 4)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane4TxAnaAtbsel2

Address

0xE981121B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.138 TX POWER STATE Control Register (Lane 4)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane4TxAnaControl

Address

0xE981121B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.139 Transmit Control Inputs Status Register (Lane 5)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane5TxStat

Address

0xE98112808

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.140 Receiver Control Inputs Status Register (Lane 5)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane5RxStat

Address

0xE98112810

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.141 Output Signals Status Register (Lane 5)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane5OutStat

Address

0xE98112818

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.142 Transmitter Control Inputs Override Register (Lane 5)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane5TxOvrd

Address

0xE98112820

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.143 Receiver Control Inputs Override Register (Lane 5)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane5RxOvrd

Address

0xE98112828

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.144 Output Signals Override Register (Lane 5)

Description

Override of output signals

Register

R_PciePhyCrLane5OutOvrd

Address

0xE98112830

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.145 Debug Control Register (Lane 5)

Description

Debug control register

Register

R_PciePhyCrLane5DbgCtl

Address

0xE98112838

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.146 Pattern Generator Controls Register (Lane 5)

Description

Pattern Generator controls

Register

R_PciePhyCrLane5PgCtl

Address

0xE98112880

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.147 Pattern Matcher Controls Register (Lane 5)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane5PmCtl

Address

0xE981128C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.148 Pattern Match Error Counter Register (Lane 5)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane5PmErr

Address

0xE981128C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.149 Current Phase Selector Value. Register (Lane 5)

Description

Current phase selector value.

Register

R_PciePhyCrLane5Phase

Address

0xE981128D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.150 Current Frequency Integrator Value. Register (Lane 5)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane5Freq

Address

0xE981128D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.151 Scope Control Register (Lane 5)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane5ScopeCtl

Address

0xE981128E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.152 Recovered Domain Receiver Control Register (Lane 5)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane5RxCtl

Address

0xE981128E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.153 Receiver Debug Register (Lane 5)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane5RxDbg

Address

0xE981128F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.154 RX Control Register (Lane 5)

Description

RX Control Bits

Register

R_PciePhyCrLane5RxAnaCtrl

Address

0xE98112980

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.155 RX ATB Register (Lane 5)

Description

RX ATB bits

Register

R_PciePhyCrLane5RxAnaAtb

Address

0xE98112988

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.156 8 Bit Programming Register (Lane 5)

Description

8 bit programming register

Register

R_PciePhyCrLane5PllPrg2

Address

0xE98112990

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.157 10 Bit Programming Register (Lane 5)

Description

10 bit programming register

Register

R_PciePhyCrLane5PllPrg1

Address

0xE98112998

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.158 10 Bit Programming Register (Lane 5)

Description

10 bit programming register

Register

R_PciePhyCrLane5PllMeas

Address

0xE981129A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.159 TX ATB Control Register (Set 1) (Lane 5)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane5TxAnaAtbsel1

Address

0xE981129A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.160 TX ATB Control Register (Set 2) (Lane 5)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane5TxAnaAtbsel2

Address

0xE981129B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.161 TX POWER STATE Control Register (Lane 5)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane5TxAnaControl

Address

0xE981129B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.162 Transmit Control Inputs Status Register (Lane 6)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane6TxStat

Address

0xE98113008

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.163 Receiver Control Inputs Status Register (Lane 6)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane6RxStat

Address

0xE98113010

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.164 Output Signals Status Register (Lane 6)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane6OutStat

Address

0xE98113018

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.165 Transmitter Control Inputs Override Register (Lane 6)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane6TxOvrd

Address

0xE98113020

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.166 Receiver Control Inputs Override Register (Lane 6)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane6RxOvrd

Address

0xE98113028

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.167 Output Signals Override Register (Lane 6)

Description

Override of output signals

Register

R_PciePhyCrLane6OutOvrd

Address

0xE98113030

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.168 Debug Control Register (Lane 6)

Description

Debug control register

Register

R_PciePhyCrLane6DbgCtl

Address

0xE98113038

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.169 Pattern Generator Controls Register (Lane 6)

Description

Pattern Generator controls

Register

R_PciePhyCrLane6PgCtl

Address

0xE98113080

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.170 Pattern Matcher Controls Register (Lane 6)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane6PmCtl

Address

0xE981130C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.171 Pattern Match Error Counter Register (Lane 6)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane6PmErr

Address

0xE981130C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.172 Current Phase Selector Value. Register (Lane 6)

Description

Current phase selector value.

Register

R_PciePhyCrLane6Phase

Address

0xE981130D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.173 Current Frequency Integrator Value. Register (Lane 6)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane6Freq

Address

0xE981130D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.174 Scope Control Register (Lane 6)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane6ScopeCtl

Address

0xE981130E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.175 Recovered Domain Receiver Control Register (Lane 6)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane6RxCtl

Address

0xE981130E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.176 Receiver Debug Register (Lane 6)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane6RxDbg

Address

0xE981130F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.177 RX Control Register (Lane 6)

Description

RX Control Bits

Register

R_PciePhyCrLane6RxAnaCtrl

Address

0xE98113180

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.178 RX ATB Register (Lane 6)

Description

RX ATB bits

Register

R_PciePhyCrLane6RxAnaAtb

Address

0xE98113188

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.179 8 Bit Programming Register (Lane 6)

Description

8 bit programming register

Register

R_PciePhyCrLane6PllPrg2

Address

0xE98113190

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.180 10 Bit Programming Register (Lane 6)

Description

10 bit programming register

Register

R_PciePhyCrLane6PllPrg1

Address

0xE98113198

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.181 10 Bit Programming Register (Lane 6)

Description

10 bit programming register

Register

R_PciePhyCrLane6PllMeas

Address

0xE981131A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.182 TX ATB Control Register (Set 1) (Lane 6)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane6TxAnaAtbsel1

Address

0xE981131A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.183 TX ATB Control Register (Set 2) (Lane 6)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane6TxAnaAtbsel2

Address

0xE981131B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.184 TX POWER STATE Control Register (Lane 6)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane6TxAnaControl

Address

0xE981131B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.185 Transmit Control Inputs Status Register (Lane 7)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrLane7TxStat

Address

0xE98113808

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	RS	X		Always reads as 1.
14:13	TxEdgerate	RS	X		Edgerate control.
12:10	TxAtten	RS	X		Attenuation amount control.
9:6	TxBoost	RS	X		Boost amount control.
5	Reserved	RS	X		Always reads as 0.
4	TxClkAlign	RS	X		Command to align clocks.
3:1	TxEn	RS	X		Transmit enable control.
0	TxCkoEn	RS	X		Tx_cko clock enable.

13.14.186 Receiver Control Inputs Status Register (Lane 7)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrLane7RxStat

Address

0xE98113810

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	RS	X		Always reads as 1.
13:12	LosCtl	RS	X		LOS filtering mode control.
11	DpllReset	RS	X		DPLL reset control.
10:8	RxDpllMode	RS	X		DPLL mode control.
7:5	RxEqVal	RS	X		Equalization amount control.
4	RxTermEn	RS	X		Receiver termination enable.
3	RxAlignEn	RS	X		Receiver alignment enable.
2	RxEn	RS	X		Receiver enable control.
1	RxPllPwron	RS	X		PLL power state control.
0	HalfRate	RS	X		Digital half-rate data control.

13.14.187 Output Signals Status Register (Lane 7)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrLane7OutStat

Address

0xE98113818

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	RS	X		Always reads as 1.
4	TxRxpres	RS	X		Transmit receiver detection result.
3	TxDone	RS	X		Transmit operation is complete output.
2	Los	RS	X		Loss of signal output.
1	RxPllState	RS	X		Current state of Rx PLL.
0	RxValid	RS	X		Receiver valid output.

13.14.188 Transmitter Control Inputs Override Register (Lane 7)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrLane7TxOvrd

Address

0xE98113820

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	RWS	0		Enable override of all bits in this register.
14:13	TxEdgerate	RW	0x0		Edgerate control.
12:10	TxAtten	RW	0x0		Attenuation amount control.
9:6	TxBoost	RW	0x0		Boost amount control.
5	Reserved	RW	0		No effect.
4	TxClkAlign	RW	0		Command to align clocks.
3:1	TxEn	RW	0x3		Transmit enable control.
0	TxCkoEn	RW	1		Tx_cko clock enable.

13.14.189 Receiver Control Inputs Override Register (Lane 7)

Description

Override of Receiver control inputs

Register

R_PciePhyCrLane7RxOvrd

Address

0xE98113828

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	RWS	0		Enable override of all bits in this register.
13:12	LosCtl	RW	0x1		LOS filtering mode control.
11	DpllReset	RW	0		DPLL reset control.
10:8	RxDpllMode	RW	0x4		DPLL mode control.
7:5	RxEqVal	RW	0x0		Equalization amount control.
4	RxTermEn	RW	1		Receiver termination enable.
3	RxAlignEn	RW	1		Receiver alignment enable.
2	RxEn	RW	1		Receiver enable control.
1	RxPllPwron	RW	1		PLL power state control.
0	HalfRate	RW	0		Digital half-rate data control.

13.14.190 Output Signals Override Register (Lane 7)

Description

Override of output signals

Register

R_PciePhyCrLane7OutOvrd

Address

0xE98113830

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	RWS	0		Enable override of all bits in this register.
4	TxRxpres	RW	1		Transmit receiver detection result.
3	TxDone	RW	0		Transmit operation is complete output.
2	Los	RW	0		Loss of signal output.
1	RxPllState	RW	0		Current state of Rx PLL.
0	RxValid	RW	1		Receiver valid output.

13.14.191 Debug Control Register (Lane 7)

Description

Debug control register

Register

R_PciePhyCrLane7DbgCtl

Address

0xE98113838

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	RW	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	RW	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	RW	0		Disable rx_ck output.
3	InvertRx	RW	0		Invert receive data (pre-lbert).
2	InvertTx	RW	0		Invert transmit data (post-lbert).
1	ZeroRxData	RW	0		Override all receive data to zeros.
0	ZeroTxData	RW	0		Override all transmit data to zeros.

13.14.192 Pattern Generator Controls Register (Lane 7)

Description

Pattern Generator controls

Register

R_PciePhyCrLane7PgCtl

Address

0xE98113880

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	RW	0x0		Pattern for modes 3-5.
3	TriggerErr	RW	0		Insert a single error into a lsb.
2:0	Mode	RW	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.193 Pattern Matcher Controls Register (Lane 7)

Description

Pattern Matcher controls

Register

R_PciePhyCrLane7PmCtl

Address

0xE981138C0

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	RW	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	RW	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.194 Pattern Match Error Counter Register (Lane 7)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrLane7PmErr

Address

0xE981138C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	RWS	X		If active, multiply COUNT by 128.
14:0	Count	RWS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.195 Current Phase Selector Value. Register (Lane 7)

Description

Current phase selector value.

Register

R_PciePhyCrLane7Phase

Address

0xE981138D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	RWS	0x0		Current phase selector value.
0	Dthr	RWS	0		Current phase selector value.

13.14.196 Current Frequency Integrator Value. Register (Lane 7)

Description

Current frequency integrator value.

Register

R_PciePhyCrLane7Freq

Address

0xE981138D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	RWS	0x0		Current frequency integrator value.
0	Dthr	RWS	0		Current frequency integrator value.

13.14.197 Scope Control Register (Lane 7)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrLane7ScopeCtl

Address

0xE981138E0

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	RW	0x0		Which bit to sample when MODE = 1.
10:2	Delay	RW	0x0		Number of symbols to skip between samples.
1:0	Mode	RW	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.198 Recovered Domain Receiver Control Register (Lane 7)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrLane7RxCtl

Address

0xE981138E8

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	RW	0		Value to override the data/phase mux.
13	OvrdSwitch	RW	0		Override the value of the data/phase mux.
12:10	ModeBp	RW	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	RW	0x0		Override value for FRUG.
7:6	PhugValue	RW	0x0		Override value for PHUG.
5	OvrdDpllGain	RW	0		Override PHUG and FRUG values.
4	PhdetPol	RW	0		Reverse polarity of phase error.
3:2	PhdetEdge	RW	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	RW	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.199 Receiver Debug Register (Lane 7)

Description

Control bits for receiver debug

Register

R_PciePhyCrLane7RxDbg

Address

0xE981138F0

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	RW	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	RW	0x0		Select wire to go on DTB bit 0.

13.14.200 RX Control Register (Lane 7)

Description

RX Control Bits

Register

R_PciePhyCrLane7RxAnaCtrl

Address

0xE98113980

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	RW	1		Unused.
4	RxlbiEn	RW	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	RW	0		Wafer level (external) loopback enable bit.
2	Rck625En	RW	0		Rck625 enable bit.
1	MarginEn	RW	0		Margin enable bit.
0	AtbEn	RW	0		ATB enable bit.

13.14.201 RX ATB Register (Lane 7)

Description

RX ATB bits

Register

R_PciePhyCrLane7RxAnaAtb

Address

0xE98113988

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	RW	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	RW	0		Connect atb_s_m to RX vcm.
3	SensemRxM	RW	0		Connect atb_s_m to rx_m.
2	SensepRxP	RW	0		Connect atb_s_p to rx_p.
1	ForcepRxM	RW	0		Connect atb_f_p to rx_m.
0	ForcepRxP	RW	0		Connect atb_f_p to rx_p.

13.14.202 8 Bit Programming Register (Lane 7)

Description

8 bit programming register

Register

R_PciePhyCrLane7PllPrg2

Address

0xE98113990

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	RW	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	RW	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	RW	0		1=force coupling in vco to maximum.
4	FrcPwron	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	RW	0		1=power is supplied to the PLL.
2	FrcReset	RW	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	RW	0		1=PLL is held/placed in reset.
0	EnableTestPd	RW	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.203 10 Bit Programming Register (Lane 7)

Description

10 bit programming register

Register

R_PciePhyCrLane7PllPrg1

Address

0xE98113998

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	RW	1		Unused.
8	SelRxck	RW	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	RW	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	RW	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	RW	0x1		Unused.

13.14.204 10 Bit Programming Register (Lane 7)

Description

10 bit programming register

Register

R_PciePhyCrLane7PllMeas

Address

0xE981139A0

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	RW	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	RW	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	RW	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	RW	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	RW	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	RW	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	RW	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	RW	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	RW	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	RW	0		Unused.

13.14.205 TX ATB Control Register (Set 1) (Lane 7)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane7TxAnaAtbsel1

Address

0xE981139A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	RW	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	RW	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	RW	0		Txm connected to ATB_S_P For term.
4	TxpSP	RW	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	RW	0		Txp connected to ATB_F_P For term.
2	VregSM	RW	0		Reg.
1	VrefSP	RW	0		Tx_vref.
0	VgrSP	RW	0		Reg.

13.14.206 TX ATB Control Register (Set 2) (Lane 7)

Description

TX ATB Control Bits

Register

R_PciePhyCrLane7TxAnaAtbsel2

Address

0xE981139B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	RW	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	RW	0		Ref.
5	VcmSP	RW	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	RW	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	RW	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	RW	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	RW	0		Enable TX internal loopback.

13.14.207 TX POWER STATE Control Register (Lane 7)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrLane7TxAnaControl

Address

0xE981139B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	RW	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	RW	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	RW	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	RW	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	RW	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	RW	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	RW	0		Unused reg.

13.14.208 PHY Reset Register

Description

Write to a 1 to reset Phy Write-only (not a real register).

Register

R_PciePhyCrReset

Address

0xE9813F9F8

Bit	Mnemonic	Access	Reset	Type	Definition
0	Reset	WS	0		Write to a 1 to reset Phy Write-only (not a real register).

13.14.209 Transmit Control Inputs Status Register (Broadcast)

Description

Status of Transmit control inputs Reset value depends on inputs

Register

R_PciePhyCrBcastTxStat

Address

0xE98151808

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Reserved1	WS	X		Always reads as 1.
14:13	TxEdgerate	WS	X		Edgerate control.
12:10	TxAtten	WS	X		Attenuation amount control.
9:6	TxBoost	WS	X		Boost amount control.
5	Reserved	WS	X		Always reads as 0.
4	TxClkAlign	WS	X		Command to align clocks.
3:1	TxEn	WS	X		Transmit enable control.
0	TxCkoEn	WS	X		Tx_cko clock enable.

13.14.210 Receiver Control Inputs Status Register (Broadcast)

Description

Status of Receiver control inputs Reset value depends on inputs

Register

R_PciePhyCrBcastRxStat

Address

0xE98151810

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
14	Reserved	WS	X		Always reads as 1.
13:12	LosCtl	WS	X		LOS filtering mode control.
11	DpllReset	WS	X		DPLL reset control.
10:8	RxDpllMode	WS	X		DPLL mode control.
7:5	RxEqVal	WS	X		Equalization amount control.
4	RxTermEn	WS	X		Receiver termination enable.
3	RxAlignEn	WS	X		Receiver alignment enable.
2	RxEn	WS	X		Receiver enable control.
1	RxPllPwron	WS	X		PLL power state control.
0	HalfRate	WS	X		Digital half-rate data control.

13.14.211 Output Signals Status Register (Broadcast)

Description

Status of output signals Reset value depends on inputs

Register

R_PciePhyCrBcastOutStat

Address

0xE98151818

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Reserved	WS	X		Always reads as 1.
4	TxRxpres	WS	X		Transmit receiver detection result.
3	TxDone	WS	X		Transmit operation is complete output.
2	Los	WS	X		Loss of signal output.
1	RxPllState	WS	X		Current state of Rx PLL.
0	RxValid	WS	X		Receiver valid output.

13.14.212 Transmitter Control Inputs Override Register (Broadcast)

Description

Override of Transmitter control inputs

Register

R_PciePhyCrBcastTxOvrd

Address

0xE98151820

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ovrd	WS	0		Enable override of all bits in this register.
14:13	TxEdgerate	WS	0x0		Edgerate control.
12:10	TxAtten	WS	0x0		Attenuation amount control.
9:6	TxBoost	WS	0x0		Boost amount control.
5	Reserved	WS	0		No effect.
4	TxClkAlign	WS	0		Command to align clocks.
3:1	TxEn	WS	0x3		Transmit enable control.
0	TxCkoEn	WS	1		Tx_cko clock enable.

13.14.213 Receiver Control Inputs Override Register (Broadcast)

Description

Override of Receiver control inputs

Register

R_PciePhyCrBcastRxOvrd

Address

0xE98151828

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
14	Ovrd	WS	0		Enable override of all bits in this register.
13:12	LosCtl	WS	0x1		LOS filtering mode control.
11	DpllReset	WS	0		DPLL reset control.
10:8	RxDpllMode	WS	0x4		DPLL mode control.
7:5	RxEqVal	WS	0x0		Equalization amount control.
4	RxTermEn	WS	1		Receiver termination enable.
3	RxAlignEn	WS	1		Receiver alignment enable.
2	RxEn	WS	1		Receiver enable control.
1	RxPllPwron	WS	1		PLL power state control.
0	HalfRate	WS	0		Digital half-rate data control.

13.14.214 Output Signals Override Register (Broadcast)

Description

Override of output signals

Register

R_PciePhyCrBcastOutOvrd

Address

0xE98151830

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Ovrd	WS	0		Enable override of all bits in this register.
4	TxRxpres	WS	1		Transmit receiver detection result.
3	TxDone	WS	0		Transmit operation is complete output.
2	Los	WS	0		Loss of signal output.
1	RxPllState	WS	0		Current state of Rx PLL.
0	RxValid	WS	1		Receiver valid output.

13.14.215 Debug Control Register (Broadcast)

Description

Debug control register

Register

R_PciePhyCrBcastDbgCtl

Address

0xE98151838

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
14:10	DtbSel1	WS	0x0		Select of wire to drive onto DTB bit 1 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
9:5	DtbSel0	WS	0x0		Select of wire to drive onto DTB bit 0 0 - disabled 1 - half_rate 2 - tx_en[0] 3.
4	DisableRxCk	WS	0		Disable rx_ck output.
3	InvertRx	WS	0		Invert receive data (pre-lbert).
2	InvertTx	WS	0		Invert transmit data (post-lbert).
1	ZeroRxData	WS	0		Override all receive data to zeros.
0	ZeroTxData	WS	0		Override all transmit data to zeros.

13.14.216 Pattern Generator Controls Register (Broadcast)

Description

Pattern Generator controls

Register

R_PciePhyCrBcastPgCtl

Address

0xE98151880

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:4	Pat0	WS	0x0		Pattern for modes 3-5.
3	TriggerErr	WS	0		Insert a single error into a lsb.
2:0	Mode	WS	0x0		Pattern to generate 0 - disabled 1 - lfsr15.

13.14.217 Pattern Matcher Controls Register (Broadcast)

Description

Pattern Matcher controls

Register

R_PciePhyCrBcastPmCtl

Address

0xE981518C0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
3	Sync	WS	0		Synchronize pattern matcher LFSR with incoming data must be turned on then off t.
2:0	Mode	WS	0x0		Pattern to match 0 - disabled 1 - lfsr15 2 - lfsr7 3 - d[n] = d[n-10] 4 - d[n] =.

13.14.218 Pattern Match Error Counter Register (Broadcast)

Description

Pattern match error counter A read resets the register. When the clock to the error counter is off, reads and writes to the register are queued until the clock is turned back on

Register

R_PciePhyCrBcastPmErr

Address

0xE981518C8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
15	Ov14	WS	X		If active, multiply COUNT by 128.
14:0	Count	WS	X		Current error count If OV14 field is active, then multiply count by 128.

13.14.219 Current Phase Selector Value. Register (Broadcast)

Description

Current phase selector value.

Register

R_PciePhyCrBcastPhase

Address

0xE981518D0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
10:1	Val	WS	0x0		Current phase selector value.
0	Dthr	WS	0		Current phase selector value.

13.14.220 Current Frequency Integrator Value. Register (Broadcast)

Description

Current frequency integrator value.

Register

R_PciePhyCrBcastFreq

Address

0xE981518D8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
13:1	Val	WS	0x0		Current frequency integrator value.
0	Dthr	WS	0		Current frequency integrator value.

13.14.221 Scope Control Register (Broadcast)

Description

Control bits for per-transceiver scope portion

Register

R_PciePhyCrBcastScopeCtl

Address

0xE981518E0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
14:11	Base	WS	0x0		Which bit to sample when MODE = 1.
10:2	Delay	WS	0x0		Number of symbols to skip between samples.
1:0	Mode	WS	0x0		Mode of counters 0 = off 1 = sample every 10 bits (see BASE) 2 = sample every 11.

13.14.222 Recovered Domain Receiver Control Register (Broadcast)

Description

Control bits for receiver in recovered domain

Register

R_PciePhyCrBcastRxCtl

Address

0xE981518E8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
14	SwitchVal	WS	0		Value to override the data/phase mux.
13	OvrdSwitch	WS	0		Override the value of the data/phase mux.
12:10	ModeBp	WS	0x0		Set BP 2:0 to longer timescale (for FTS patterns) BP0 - Start PHUG profile at 4/.
9:8	FrugValue	WS	0x0		Override value for FRUG.
7:6	PhugValue	WS	0x0		Override value for PHUG.
5	OvrdDpllGain	WS	0		Override PHUG and FRUG values.
4	PhdetPol	WS	0		Reverse polarity of phase error.
3:2	PhdetEdge	WS	0x3		Edges to use for phase detection top bit is rising edges, bottom is falling.
1:0	PhdetEn	WS	0x3		Enable phase detector top bit is odd slicers, bottom is even.

13.14.223 Receiver Debug Register (Broadcast)

Description

Control bits for receiver debug

Register

R_PciePhyCrBcastRxDbg

Address

0xE981518F0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7:4	DtbSel1	WS	0x0		Select wire to go on DTB bit 1.
3:0	DtbSel0	WS	0x0		Select wire to go on DTB bit 0.

13.14.224 RX Control Register (Broadcast)

Description

RX Control Bits

Register

R_PciePhyCrBcastRxAnaCtrl

Address

0xE98151980

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	Unused	WS	1		Unused.
4	RxlbiEn	WS	0		Digital serial (internal) loopback enable bit.
3	RxlbeEn	WS	0		Wafer level (external) loopback enable bit.
2	Rck625En	WS	0		Rck625 enable bit.
1	MarginEn	WS	0		Margin enable bit.
0	AtbEn	WS	0		ATB enable bit.

13.14.225 RX ATB Register (Broadcast)

Description

RX ATB bits

Register

R_PciePhyCrBcastRxAnaAtb

Address

0xE98151988

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
5	SensemVrefLos	WS	0		Connect atb_s_m to vref_los (vref_rx/14).
4	SensemVcm	WS	0		Connect atb_s_m to RX vcm.
3	SensemRxM	WS	0		Connect atb_s_m to rx_m.
2	SensepRxP	WS	0		Connect atb_s_p to rx_p.
1	ForcepRxM	WS	0		Connect atb_f_p to rx_m.
0	ForcepRxP	WS	0		Connect atb_f_p to rx_p.

13.14.226 8 Bit Programming Register (Broadcast)

Description

8 bit programming register

Register

R_PciePhyCrBcastPllPrg2

Address

0xE98151990

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbSenseSel	WS	0		Control of Proportional charge pump current 1=Enable signals internal to the PLL.
6	FrcHcpl	WS	0		Allow override of default value of hcpl 1=allow hcpl_lcl to control high-couplin.
5	HcplLcl	WS	0		1=force coupling in vco to maximum.
4	FrcPwron	WS	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
3	PwronLcl	WS	0		1=power is supplied to the PLL.
2	FrcReset	WS	0		Allow override of default value of pll_pwron 1=allow pwron_lcl to control pll po.
1	ResetLcl	WS	0		1=PLL is held/placed in reset.
0	EnableTestPd	WS	0		1=phase linearity of phase interpolator and VCO is being tested.

13.14.227 10 Bit Programming Register (Broadcast)

Description

10 bit programming register

Register

R_PciePhyCrBcastPllPrg1

Address

0xE98151998

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	Unused1	WS	1		Unused.
8	SelRxck	WS	0		Use recovered clock as reference to the PLL.
7:5	PropCntrl	WS	0x5		Control of Proportional charge pump current Proportional current = (n+1)/8*full_.
4:2	IntCntrl	WS	0x2		Control of Integral charge pump current Integral current = (n+1)/8*full_scale De.
1:0	Unused	WS	0x1		Unused.

13.14.228 10 Bit Programming Register (Broadcast)

Description

10 bit programming register

Register

R_PciePhyCrBcastPllMeas

Address

0xE981519A0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
9	MeasBias	WS	0		Measure copy of bias current in oscillator on atb_force_m.
8	MeasVcntrl	WS	0		Measure vcntrl on atb_sense_m If MEAS_VREF is set as well, atb_sense_p,m mea- su.
7	MeasVref	WS	0		Measure vref on atb_sense_p; gd on atb_sense_m If MEAS_VCNTRL is set as well, at.
6	MeasVp16	WS	0		Measure vp16 on atb_sense_p; gd on atb_sense_m.
5	MeasStartup	WS	0		Measure startup voltage on atb_sense_p; gd on atb_sense_m.
4	MeasVco	WS	0		Measure vco supply voltage on atb_sense_p; gd on atb_sense_m.
3	MeasVpCp	WS	0		Measure vp_cp voltage on atb_sense_p; gd on atb_sense_m If MEAS_1V is set as wel.
2	Meas1v	WS	0		Measure 1V supply voltage on atb_sense_m If MEAS_VP_CP is set as well, atb_sense.
1	MeasCrowbar	WS	0		Measure crowbar bias voltage on atb_sense_p; gd on atb_sense_m.
0	Unused	WS	0		Unused.

13.14.229 TX ATB Control Register (Set 1) (Broadcast)

Description

TX ATB Control Bits

Register

R_PciePhyCrBcastTxAnaAtbsel1

Address

0xE981519A8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	VbpfSP	WS	0		Vbpf in edge rate control circuit on ATB_S_P Set ATB_EN to make this useful.
6	TxmSM	WS	0		Txm on ATB_S_M Set ATB_EN to make this useful.
5	TxmFP	WS	0		Txm connected to ATB_S_P For term.
4	TxpSP	WS	0		Txp connected to ATB_S_P Set ATB_EN to make this useful.
3	TxpFP	WS	0		Txp connected to ATB_F_P For term.
2	VregSM	WS	0		Reg.
1	VrefSP	WS	0		Tx_vref.
0	VgrSP	WS	0		Reg.

13.14.230 TX ATB Control Register (Set 2) (Broadcast)

Description

TX ATB Control Bits

Register

R_PciePhyCrBcastTxAnaAtbsel2

Address

0xE981519B0

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	AtbEn	WS	0		Connect internal and external ATB busses Needed for all ATB measurements.
6	VrefrxdSM	WS	0		Ref.
5	VcmSP	WS	0		Vcm replica on ATB_S_P Set ATB_EN to make this useful.
4	VbnsSM	WS	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
3	VbpsSP	WS	0		Vbps in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
2	VbnfSM	WS	0		Vbnf in edge rate control circuit on ATB_S_M Set ATB_EN to make this useful.
1	Enlpbk	WS	0		Enable TX external loopback Make sure internal loopback is not ON.
0	EnTxilpbk	WS	0		Enable TX internal loopback.

13.14.231 TX POWER STATE Control Register (Broadcast)

Description

TX POWER STATE Control Bits

Register

R_PciePhyCrBcastTxAnaControl

Address

0xE981519B8

Attributes

-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
7	FrcPwrst	WS	0		Locally force power state tx_en<1:0> input overridden by EN_LCL.
6:5	EnLcl	WS	0x0		Locally force tx_en<1:0> 00 - power off 01 - tx idle (slow) 10 - transmit data 1.
4	FrcDo	WS	0		Force Dataovrd locally When ON, overrides input data_ovrd value.
3	DataovrdLcl	WS	0		Local dataovrd control value Set FRC_DO to make this useful.
2	FrcBeacon	WS	0		Force Beacon to local value (BCN_LCL) When On, BCN_LVL overrides input value.
1	BcnLcl	WS	0		Local Beacon On/Off Control Value Set FRC_BEACON to make this useful.
0	Unused	WS	0		Unused reg.

13.15 Transaction, Link, MAC Layers

Please reference the Synopsys' "PCI-Express Controller Core Data Book". This provides a description of the pins, the timing requirements, and the programmer-visible registers. We have configured the PCI-Express RC core to fit our use. This section documents our configuration choices.

Parameters for design DWC_pcie_rc

General Configuration

Parameter	Value	Description	Default?	Disabled?
Symbols per Cycle / Operating Frequency	1	Parameter Name : CX_NB Specifies the operating frequency of the core. This is also referred to as the number of symbols that are handled each core_clk cycle, or the S-ness of the core. 1S => 250MHz : 2S => 125MHz.	Y	
Maximum Number of Lanes Supported	8	Parameter Name : CX_NL Specifies the maximum number of lanes that are supported by the core.	Y	
Datapath Width	2	Parameter Name : CX_NW Specifies the width of the datapath (number of dwords per cycles)	Y	Y
Number of Virtual Channels	1	Parameter Name : CX_NVC Specifies the number of Virtual Channels supported by the core. A maximum of eigth VCs are supported.	Y	
Enable ECRC Support	1	Parameter Name : CX_ECRC_ENABLE Removes support for ECRC. May be disabled for smaller gate size if the Core is placed in a system where it's guaranteed that received TLPs don't contain ECRC AND the Application does not transmit ECRC from the Client interfaces. This option is only available when Include Target Interface 1 is selected.	Y	
RAM data error protection config	1	Parameter Name : CX_RAM_PROTECTION_MODE RAM data error protection mode Parity: Selects parity to check RAM data error. ECC: Selects ECC to check and correct RAM data error. None: Disables both parity and ECC modes.		
Parity Config	8	Parameter Name : CX_PAR_MODE Config RAM data width per parity bit		
RAM ECC pipeline enable	0	Parameter Name : CX_ECC_PIPE_EN Enable RAM ECC pipeline	Y	Y
Remove Port Logic Registers	0	Parameter Name : CX_PL_REG_DISABLE Removes Port Logic registers	Y	
Use RocketIO PHY	0	Parameter Name : RIO_POPULATED FPGA design using Xilinx RocketIO PHY	Y	
DBI ReadOnly Write Enable	0x1	Parameter Name : CX_DBI_RO_WR_EN Enable ReadOnly/HwInit registers to be writable through DBI		
FPGA	0	Parameter Name : FPGA This parameter specifies FPGA application	Y	
Include Target Interface 1	1	Parameter Name : TRGT1_POPULATE Specifies the inclusion or omission of the Target interface 1.	Y	Y
Application Error Reporting	0	Parameter Name : APP_RETURN_ERR_EN Determines whether to include input ports for application-detected error reporting.	Y	
Mask Completion Timeout Errors	0	Parameter Name : CPL_TIMEOUT_ERR_MASK When defined, no error will be reported to the CDM, application is responsible for returning cpl timeout error through the application error return interface ('APP_RETURN_ERR_EN).	Y	Y

Parameter	Value	Description	Default?	Disabled?
Enable Address Alignment	0	Parameter Name : GLOB_ADDR_ALIGN_EN Allows the application to enable address alignment When enabled, the core performs address alignment and generates the first and last byte enables based on the address and number of bytes of the TLP requested from the client interface. NOTE: (This note applies to all Switch Applications and to EP, RC, or Dual Mode Applications that don't have the CX_ECRC_EN macro defined): For Switch Applications and other Applications where the CX_ECRC_EN is not defined, this should normally be disabled. However, if the Application requires this to be enabled, then the address alignment pin at the top-level of the Application should only be high for those TLP's without ECRC. TLP's w/ ECRC that are being transmitted by the Application, the address alignment pin should be de-asserted for that TLP.	Y	
Provide Control to Flip Physical RX/TX Lanes	1	Parameter Name : CX_LANE_FLIP_CTRL_EN provide control allowing physical RX/TX lanes to be flipped when this feature is enabled, two pins are provided to control of RX/TX separtely. input rx_lane_flip_en; 0 – requires the RX LSB lane (i.e lane0) to be physically presented. 1 – enables flipping, of the RX MSB lane to lane0. input tx_lane_flip_en; 0 – requires the TX LSB lane (i.e lane0) to be physically presented. 1 – enables flipping, of the TX MSB lane to lane0.		
Number of Fast Training (NFTS) Sequences	15	Parameter Name : CX_NFTS Specifies the number of Fast Training Sequences the core advertises during link training. This is used to inform the link partner the cores ability to recover synchronization after a low power state. This number should come from the SerDes vendor. Legal values are in the range 1 - 255	Y	
NFTS when using common clock	15	Parameter Name : CX_COMM_NFTS Specifies the number of Fast Training Sequences the core advertises during link training when common clock configuration is set. Legal values are in the range 1 - 255	Y	
Technology Speed	0x2	Parameter Name : CX_TECHNOLOGY Specifies the speed of the technology relative to the clock frequency and architecture. This parameter is used to enable additional pipelined stages internal to the core to tradeoff latency and gates for ease of timing closure. Note: This is always SLOW for FPGA's	Y	
Disable Lane Deskew	0x0	Parameter Name : CX_DESKEW_DISABLE Enable or disable lane deskew. This should be used with care.	Y	
Enable Lane Reversal Support	0x1	Parameter Name : CX_LANE_REVERSE Enable or disable core support for lane reversal	Y	
Enable ASPM L1 Timeout	0x1	Parameter Name : CX_ASPM_TIMEOUT_ENTR_L1_EN Enable or disable the ASPM L1 timer. When enabled, core will automatically go to L1 when the timer expires and the conditions in the PCIe Specification are met.	Y	
Maximum Tags Supported	31	Parameter Name : CX_MAX_TAG Specifies the maximum number of tags supported by the core. Used to size the completion look-up-table and timeout ram.	Y	
LBC Address Bus Width	32	Parameter Name : CX_LBC_EXT_AW Specifies the width of the external Local Bus Controllers (LBC) address bus. Note: This feature is not applicable for RC.	Y	Y
Enable Diagnostic Bus	0	Parameter Name : DIAGNOSTIC_ENABLE Enables routing of important diagnostic signals out of the top level.	Y	
Maximum Payload Size Supported	512	Parameter Name : CX_MAX_MTU Specifies the maximum packet payload size supported by the core. This parameter is used to size core memories.		

Parameter	Value	Description	Default?	Disabled?
Enable Optional Checks	0	Parameter Name : ENABLE_OPTIONAL_CHECKS Adds optional protocol checks including byte enable and flow control.	Y	

RAM Configuration

Parameter	Value	Description	Default?	Disabled?
Use External RAMs	1	Parameter Name : CX_RAM_AT_TOP_IF Specifies whether to use extrnal RAMs and include top-level interface or used embedded RAMs	Y	
RAM Type	0	Parameter Name : CX_RAM_TYPE Specifies the type of ram model to use	Y	Y
RAM Timing Model	0	Parameter Name : RAM_TIMING_MODEL Specifies whether to use Black Box timing, or physical RAM timing model if black box timing is specified, Black box RAMS will be used to synthesize, and timing constraints for RAM interfaces will be derived from RAM*P_RD_ACCESS/RAM*P_ADDR_SU parameters. if physical RAM timing is specified, it is expected to be provided by your physical RAM model. [used by synth. timing model]	Y	
single port RAM Read Access Time [ps]	1249	Parameter Name : RAM1P_RD_ACCESS Specifies the single port RAM read Access time [used by synth. timing model]	Y	
single port RAM Address/Data Setup Time [ps]	893	Parameter Name : RAM1P_ADDR_SU Specifies the single port RAM data setup [used by synth. timing model]	Y	
dual port RAM Read Access Time [ps]	1444	Parameter Name : RAM2P_RD_ACCESS Specifies the dual port RAM read Access time [used by synth. timing model]	Y	
dual port RAM Address/Data Setup Time [ps]	794	Parameter Name : RAM2P_ADDR_SU Specifies the dual port RAM data setup [used by synth. timing model]	Y	

Transmit Configuration

Parameter	Value	Description	Default?	Disabled?
Include 3rd Client Interface	0	Parameter Name : CLIENT2_POPULATED Determines whether to include top-level ports for the optional third application transmit client interface (XALI2).	Y	
Block Client 0 Interface	0x1	Parameter Name : CX_CLIENT0_BLOCK_NEW_TLP This is designed to allow customer to select whether or not to allow XADM arbiter to block client0 interface When PMC is enabled with L1 and L2, L3, there will be conditions that new TLP should be blocked. But completions are always need to go. Therefore if customer configures the completion and new TLP requests combined into client0 interface, then it needs to set this value to 0 and takes over the blocking function by monitoring the output signal pm_xtlh_block_tlp. Note: If core lbc is used or one client interface is used for completions, then these block parameters should be set accordingly. For example, if client0 interface has been used for completion, then the parameter for client0 should be set to '0' so xadm arbiter will not block this interface.	Y	

Parameter	Value	Description	Default?	Disabled?
Block Client 1 Interface	0x0	Parameter Name : CX_CLIENT1_BLOCK_NEW_TLP This is designed to allow customer to select whether or not to allow XADM arbiter to block client1 interface When PMC is enabled with L1 and L2, L3, there will be conditions that new TLP should be blocked. But completions are always need to go. Therefore if customer configures the completion and new TLP requests combined into client1 interface, then it needs to set this value to 0 and takes over the blocking function by monitoring the output signal pm_xtlh_block_tlp. Note: If core lbc is used or one client interface is used for completions, then these block parameters should be set accordingly. For example, if client1 interface has been used for completion, then the parameter for client1 should be set to '0' so xadm arbiter will not block this interface.		
Block Client 2 Interface	0x1	Parameter Name : CX_CLIENT2_BLOCK_NEW_TLP This is designed to allow customer to select whether or not to allow XADM arbiter to block client1 interface When PMC is enabled with L1 and L2, L3, there will be conditions that new TLP should be blocked. But completions are always need to go. Therefore if customer configures the completion and new TLP requests combined into client1 interface, then it needs to set this value to 0 and takes over the blocking function by monitoring the output signal pm_xtlh_block_tlp. Note: If core lbc is used or one client interface is used for completions, then these block parameters should be set accordingly. For example, if client1 interface has been used for completion, then the parameter for client1 should be set to '0' so xadm arbiter will not block this interface.	Y	Y
Populate ports for available credit buses	0	Parameter Name : XADM_CRD_EN This parameter enables the population of output ports for application monitoring of run-time Avaliable credit information for VCn buses: xadm_ph_cdts [NVC*8-1:0] : available VC0 - VCn header posted credits xadm_nph_cdts [NVC*8-1:0] : available VC0 - VCn header non-posted credits xadm_cplh_cdts [NVC*8-1:0] : available VC0 - VCn header completion credits xadm_pd_cdts [NVC*12-1:0] : available VC0 - VCn data posted credits xadm_npd_cdts [NVC*12-1:0] : available VC0 - VCn data non-posted credits xadm_cpld_cdts [NVC*12-1:0] : available VC0 - VCn data completion credits Informatin for lower order VCs is presented on the lower-order bits.	Y	

Transmit Arbitration

Parameter	Value	Description	Default?	Disabled?
Transmit Arbitration Method	1	Parameter Name : CX_XADM_ARB_MODE Transmit Arbitration Method Client-Based: Provides Round Robin Arbitration Priority, among transmit clients. Strict Pri.: Provides Strict Arbitration Priority, among transmit clients. Client 0 has the lowest priority. VC-Based: (available 5/2005) Provides VC based Arbitration Priority across 2 VC classes LPVC/HPVC - LPVC groups can be programmed to render Weighted Round Robin or Round Robin Priority - HPVC groups provide Strict priority Arbitration, with priority toward highest VIDs.	Y	Y
Client Interface TLP pullback feature	0	Parameter Name : CLIENT_PULLBACK When enabled, the client interfaces are allowed to cancel a TLP currently submitted for transmission	Y	

Parameter	Value	Description	Default?	Disabled?
Enable LPVC WRR Weights Writable	0	Parameter Name : CX_LPVC_WRR_WEIGHT_WRITABLE Enable LPVC Weighted Round Robin Weights registers to be writable through DBI	Y	Y
VC ID #0 Weight	0xf	Parameter Name : LPVC_WRR_WEIGHT_VC0 WRR Weighting for VC ID #0	Y	Y
VC ID #1 Weight	0x0	Parameter Name : LPVC_WRR_WEIGHT_VC1 WRR Weighting for VC ID #1	Y	Y
VC ID #2 Weight	0x0	Parameter Name : LPVC_WRR_WEIGHT_VC2 WRR Weighting for VC ID #2	Y	Y
VC ID #3 Weight	0x0	Parameter Name : LPVC_WRR_WEIGHT_VC3 WRR Weighting for VC ID #3	Y	Y
VC ID #4 Weight	0x0	Parameter Name : LPVC_WRR_WEIGHT_VC4 WRR Weighting for VC ID #4	Y	Y
VC ID #5 Weight	0x0	Parameter Name : LPVC_WRR_WEIGHT_VC5 WRR Weighting for VC ID #5	Y	Y
VC ID #6 Weight	0x0	Parameter Name : LPVC_WRR_WEIGHT_VC6 WRR Weighting for VC ID #6	Y	Y
VC ID #7 Weight	0x0	Parameter Name : LPVC_WRR_WEIGHT_VC7 WRR Weighting for VC ID #7	Y	Y

XADMPosted

Parameter	Value	Description	Default?	Disabled?
Special Posted TLP Handling	0	Parameter Name : SPECIAL_MAX_P_CRD_ENABLE This parameter enables user to specify the necessary credits accumulated before core will transmit Posted TLPs. Note: This option cannot be selected if Compare Posted Credit is selected.	Y	
Posted TLP Credit Threshold	32	Parameter Name : SPECIAL_MAX_P_CRD This parameter defines the actual amount of Posted TLP credits core must accumulate before transmitting a posted TLP	Y	Y
Compare Posted Credit	0	Parameter Name : P_LEN_CMP_ENABLE This parameter enables core to compare the requested posted payload length against enough accumulated credits before transmission Note: This option cannot be selected if Special Posted TLP Handling is selected.	Y	

Transmit Completion

Parameter	Value	Description	Default?	Disabled?
Special Completion Handling	0	Parameter Name : SPECIAL_MAX_CPL_CRD_ENABLE This parameter enables user to specify the necessary credits accumulated before core will transmit the completions. Note: This option cannot be selected if Compare Completion Credit is selected.	Y	
Completion Credit Threshold	32	Parameter Name : SPECIAL_MAX_CPL_CRD This parameter defines the actual amount of completion credits core must accumulate before transmitting a completion	Y	Y
Compare Completion Credit	0	Parameter Name : CPL_LEN_CMP_ENABLE This parameter enables core to compare the requested completion length against enough accumulated credits before transmission Note: This option cannot be selected if Special Completion Handling is selected.	Y	

Common Register Configuration

Application Interface Options

Parameter	Value	Description	Default?	Disabled?
Configuration Upper Limit	0x3ff	Parameter Name : CONFIG_LIMIT Upper limit of internally handled Configuration requests. Any access to configuration register above this address will go to TRGT1 interface	Y	Y
Default Target Interface	0x0	Parameter Name : DEFAULT_TARGET Target Interface Destination for received TLPs which are Unsupported Requests Note: This feature is not applicable for RC.	Y	Y
Target CPL LUT Enable	0	Parameter Name : TRGT_CPL_LUT_EN Let the core calculate the correct byte count for CPL of incoming MemRd This feature is available only if target 1 Interface is included. Note: This feature is not available for Switch.	Y	
Maximum Remote Tags Supported	31	Parameter Name : CX_REMOTE_MAX_TAG Specifies the maximum number of tags track in the Target Completion LUT Used to size the target completion look-up-table and timeout ram.	Y	Y
RADM CPL LUT STORE BYTE COUNT	0	Parameter Name : RADM_CPL_LUT_STORE_BYTE_CNT Store the byte count in the RADM completion LUT	Y	
Client Data/Address Bus Parity Protection	0	Parameter Name : CX_CLIENT_PAR_MODE Select client address/data parity mode.	Y	
Application Par Error Out Enable	0	Parameter Name : APP_PAR_ERR_OUT_EN Allow application to monitor parity errors from core RAMs.	Y	
Application Return CRD Enable	0	Parameter Name : APP_RETURN_CRD_EN Allow application to directly control credit returns for each packet type.	Y	

Port Logic Register

Parameter	Value	Description	Default?	Disabled?
Default Link Number	0x4	Parameter Name : DEFAULT_LINK_NUM Default Link Number value that the EP Core advertises to the Link partner. Valid values are 0-255. (Only in RC/SW_DOWN mode)	Y	
Default ACK Frequency	0x0	Parameter Name : DEFAULT_ACK_FREQUENCY The EP Core accumulates the number of pending Ack's specified here (up to 255) before sending an Ack.	Y	
Default Replay Timer Adjustment	0x1	Parameter Name : DEFAULT_REPLAY_ADJ Default replay timer adjustment. Each value increase the replay timer by 64.	Y	
Default L1 Entry Latency	0x2	Parameter Name : DEFAULT_L1_ENTR_LATENCY L1 Entrance Latency	Y	
Default L0S Entry Latency	0x3	Parameter Name : DEFAULT_L0S_ENTR_LATENCY L0s Entrance Latency	Y	

MSI/MSI-X

Parameter	Value	Description	Default?	Disabled?
MSI Capability	0x1	Parameter Name : MSI_CAP_ENABLE MSI Capability structure enable	Y	
Enable 64-bit MSI Support	0x1	Parameter Name : MSI_64_EN 64-bit address MSI enable	Y	Y
Default Multiple MSI Capability	0x0	Parameter Name : DEFAULT_MULTI_MSI_CAPABLE Indicates that multiple Message mode is enabled by system software. The number of Messages enabled must be less than or equal to the Multiple Message Capable value.	Y	
MSI-X Capability	0x0	Parameter Name : MSIX_CAP_ENABLE MSI-X Capability enable	Y	

PCIe Capability

Parameter	Value	Description	Default?	Disabled?
L0S Exit Latency	0x3	Parameter Name : DEFAULT_L0S_EXIT_LATENCY L0s Exit Latency	Y	
L0S Exit Latency (common clk)	0x3	Parameter Name : DEFAULT_COMM_L0S_EXIT_LATENCY L0s Exit Latency when using common clock	Y	
L1 Exit Latency	0x6	Parameter Name : DEFAULT_L1_EXIT_LATENCY L1 Exit Latency	Y	
L1 Exit Latency (common clk)	0x6	Parameter Name : DEFAULT_COMM_L1_EXIT_LATENCY L1 Exit Latency when using common clock	Y	
Port Number	0x0	Parameter Name : PORT_NUM PCIe Port number for the given PCIe link	Y	
Use Platform Reference Clock	0x1	Parameter Name : SLOT_CLK_CONFIG Slot Clock Configuration Indicates that the component uses the same physical reference clock that the platform provides on the connector.	Y	
Physical Slot Number	0x0	Parameter Name : SLOT_PHY_SLOT_NUM Physical Slot Number	Y	
Slot Power Limit Scale	0x0	Parameter Name : SET_SLOT_PWR_LIMIT_SCALE Slot Power Limit Scale - Specifies the scale used for the Slot Power Limit Value	Y	
Slot Power Limit Value	0xf	Parameter Name : SET_SLOT_PWR_LIMIT_VAL Slot Power Limit Value - Upper limit of power supplied by slot		
Slot is Hot-Plug Capable	0x1	Parameter Name : SLOT_HP_CAPABLE When set indicates that this slot is capable of supporting Hot-Plug operations		
Slot Support Hot-Plug Surprise	0x1	Parameter Name : SLOT_HP_SURPRISE When set indicates that a device present in this slot might be removed from the system without any prior notification		
Disable Hot-Plug Software Notification	0x0	Parameter Name : SLOT_NO_CC_SUPPORT When set, it indicates that this slot doesn't generate software notification when an issued command is completed by the Hot-Plug Controller	Y	
Electro-mechanical Interlock Implemented	0x0	Parameter Name : SLOT_EML_PRESENT When set, it indicates that an Electromechanical Interlock is implemented on the chassis for this slot.	Y	
Slot Power Indicator Present	0x1	Parameter Name : SLOT_PWR_IND_PRESENT When set indicates that a Power Indicator is implemented on the chassis for this slot.		
Slot Attention Indicator Present	0x1	Parameter Name : SLOT_ATTEN_IND_PRESENT When set indicates that an Attention Indicator is implemented on the chassis for this slot.		
Slot MRL Sensor Present	0x0	Parameter Name : SLOT_MRL_SENSOR_PRESENT When set indicates that an MRL Sensor is implemented on the chassis for this slot.	Y	
Slot Power Controller Present	0x1	Parameter Name : SLOT_PWR_CTRL_PRESENT When set indicates that a Power Controller is implemented for this slot.		
Slot Attention Button Present	0x0	Parameter Name : SLOT_ATTEN_BUTTON_PRESENT When set indicates that an Attention Button is implemented on the chassis for this slot.	Y	

PCIe Extended Capabilities

Parameter	Value	Description	Default?	Disabled?
Support Advanced Error Reporting	0x1	Parameter Name : AER_ENABLE Advanced Error Reporting Capability enable	Y	Y
Virtual Channel Support	0x0	Parameter Name : VC_ENABLE Virtual Channel Capability enable		

Parameter	Value	Description	Default?	Disabled?
Serial Number Capability	0x0	Parameter Name : SERIAL_CAP_ENABLE Device Serial Number Capability enable	Y	
Device Serial Number (1st DW)	0x0	Parameter Name : DEFAULT_SN_DW1 Specifies the first 32-bit device serial number	Y	Y
Device Serial Number (2nd DW)	0x0	Parameter Name : DEFAULT_SN_DW2 Specifies the second 32-bit device serial number	Y	Y

Vital Product Data (VPD)

Parameter	Value	Description	Default?	Disabled?
VPD Capability	0x0	Parameter Name : VPD_CAP_ENABLE Vital Product Data (VPD) Capability structure enable	Y	

Virtual Channel Capability

Parameter	Value	Description	Default?	Disabled?
VC Arbitration Capability	0x0	Parameter Name : DEFAULT_VC_ARB_32 Types of VC Arbitration supported by the device for the LPVC group bit 0 - Weighted Round Robin arbitration with 16 phases bit 1 - Weighted Round Robin arbitration with 32 phases bit 2 - Weighted Round Robin arbitration with 64 phases bit 3 - Weighted Round Robin arbitration with 128 phases bit 4-7 Reserved	Y	Y
Low Priority Extended VC Count	0x0	Parameter Name : DEFAULT_LOW_PRI_EXT_VC_CNT Indicates the number of (exteded) VC in addition to the default VC belonging to the LPVC group that has the lowest priority with respect to other VC resources in a strict-priority VC arbitration.	Y	Y

Function Configuration

Function 0 Configuration

Function 0 -> PCI Express Capability

Parameter	Value	Description	Default?	Disabled?
PCIe Capabilities Interrupt Message Number	0x0	Parameter Name : PCIE_CAP_INT_MSG_NUM_0 This register indicates which MSI/MSI-X vector is used for the interrupt message generated in association with the status bits in either the Slot Status register	Y	
Clock PM Support	0x0	Parameter Name : DEFAULT_CLK_PM_CAP_0 When set indicates that the component tolerates the removal of any ref clk when the link is in the L1 and L2/3 ready states.	Y	
Is Port Connected to Slot	0x1	Parameter Name : SLOT_IMPLEMENTED_0 When set indicates that the PCI Express Link associated with this Port is connected to a slot		
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_0 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	
Add Support For Attention Button	0x0	Parameter Name : DEFAULT_ATT_BUTT_PRE_0 When set indicates that an Attention Button is present	Y	
Add Support For Attention Indicator	0x1	Parameter Name : DEFAULT_ATT_IND_PRE_0 When set indicates that an Attention Indicator is present		

Parameter	Value	Description	Default?	Disabled?
Add Support For Power Indicator	0x1	Parameter Name : DEFAULT_PWR_IND_PRE_0 When set indicates that a Power Indicator is present		
Support No-Snoop	0x0	Parameter Name : DEFAULT_NO_SNOOP_SUPPORTED_0 When set indicates that the device is permitted to set the No Snoop bit in the Requester Attributes of transactions it initiates that do not require hardware enforced cache coherency	Y	
Active State Link PM Support	0x3	Parameter Name : AS_LINK_PM_SUPT_0 Active State Power Management Support	Y	
Enable Root RCB	0x0	Parameter Name : ROOT_RCB_0 Indicates the RCB value for the Root Port (RC-Only)	Y	

Function 0 -> MSI-X Register Configuration

Parameter	Value	Description	Default?	Disabled?
MSIX Table Size	0x0	Parameter Name : MSIX_TABLE_SIZE_0 MSI-X Table Size - Encoded as (Table Size - 1).	Y	Y
MSIX Table BIR	0x0	Parameter Name : MSIX_TABLE_BIR_0 Table BAR Indicator Register (BIR) Indicates which BAR is used to map the MSI-X Table into memory space	Y	Y
MSIX Table Offset	0x0	Parameter Name : MSIX_TABLE_OFFSET_0 Table Offset - Base address of the MSI-X Table, as an offset from the base address of the BAR indicated by the table BIR bits.	Y	Y
MSIX PBA BIR	0x0	Parameter Name : MSIX_PBA_BIR_0 Pending Bit Array (PBA) BIR Indicates which BAR is used to map the MSI-X PBA into memory space	Y	Y
MSIX PBA Offset	0x0	Parameter Name : MSIX_PBA_OFFSET_0 PBA Offset - Base address of the MSI-X PBA, as an offset from the base address of the BAR indicated by the PBA BIR bits.	Y	Y

Function 0 -> Advanced Error Register Configuration

Parameter	Value	Description	Default?	Disabled?
Default ECRC Check Capability	0x1	Parameter Name : DEFAULT_ECRC_CHK_CAP_0 ECRC Checking Capability	Y	
Default ECRC Generation Capability	0x1	Parameter Name : DEFAULT_ECRC_GEN_CAP_0 ECRC Generation Capability	Y	
Advanced Error Interrupt Message Number	0x0	Parameter Name : AER_INT_MSG_NUM_0 This register must indicate which MSI/MSI-X vector is used for the interrupt message generated in association with any of the status bits of this capability	Y	

Function 0 -> Power Management Register Configuration

Parameter	Value	Description	Default?	Disabled?
PME Support	0x1b	Parameter Name : PME_SUPPORT_0 5-bit field indicates the power states in which the device may generate a PME.	Y	
D1 Support	0x1	Parameter Name : D1_SUPPORT_0 Supports the D1 PM state	Y	
D2 Support	0x0	Parameter Name : D2_SUPPORT_0 Supports the D2 PM state	Y	
Device Specific Initialization	0x0	Parameter Name : DEV_SPEC_INIT_0 Device Specific Initialization	Y	
Auxiliary Current	0x7	Parameter Name : AUX_CURRENT_0 Auxillary Current requirement	Y	

Parameter	Value	Description	Default?	Disabled?
No Reset on D3hot->D0 Transition	0x0	Parameter Name : DEFAULT_NO_SOFT_RESET_0 When set, it indicates that this device when transitioning from D3hot to D0 because of powerstate commands don't perform an internal reset.	Y	

Function 0 -> PCI Register Configuration

Parameter	Value	Description	Default?	Disabled?
Device Identification Number	0x1	Parameter Name : CX_DEVICE_ID_0 Specifies the 16-bit device identification number for the function.		
Vendor Indentification Number	0x19b2	Parameter Name : CX_VENDOR_ID_0 Specifies the 16-bit vendor identication number for the function. This value is controlled by the PCI SIG.		
Device Revision Number	0x1	Parameter Name : CX_REVISION_ID_0 Specifies the 8-bit revision number of the function.	Y	
Base Class Code	0x6	Parameter Name : BASE_CLASS_CODE_0 Class code		
Sub Class Code	0x4	Parameter Name : SUB_CLASS_CODE_0 Sub-class code		
Programming Interface Code	0x0	Parameter Name : IF_CODE_0 Programming Interface code	Y	
IO Address Decode	0x1	Parameter Name : IO_DECODE_32_0 IO Addressing (Type1-Only) **NOTE** Should not appear for EP		
Memory Address Decode	0x1	Parameter Name : MEM_DECODE_64_0 Memory Addressing (Type1-Only) **NOTE** Should not appear for EP		
Enable ROM BAR	0x0	Parameter Name : ROM_BAR_ENABLED_0 ROM BAR Enable		
ROM BAR Mask	0xffff	Parameter Name : ROM_MASK_0 ROM BAR Mask ex: 32'hFFFF = BAR size of 2ˆ16. Set to all Fs to disable	Y	Y
Allow Reprogramming of ROM BAR Mask	0x0	Parameter Name : ROM_MASK_WRITABLE_0 When set enables dynamic changing of ROM BAR Mask through DBI	Y	Y
Specify ROM BAR Target Interface	0x1	Parameter Name : ROM_FUNC0_TARGET_MAP Destination of request matching ROM BAR Note: This feature is not applicable for RC.	Y	Y

Function 0 -> BAR_0 / BAR_1

Parameter	Value	Description	Default?	Disabled?
Enable BAR_0	0x0	Parameter Name : BAR0_ENABLED_0 BAR0 Enable		
BAR_0 is Memory or I/O	0x0	Parameter Name : MEM0_SPACE_DECODER_0 BAR0 Memory Space Indicator When set indicates IO space	Y	Y
BAR_0 is Prefetchable	0x0	Parameter Name : PREFETCHABLE0_0 BAR0 Memory Prefetchable When set indicates BAR0 Memory BAR is a prefetchable BAR	Y	Y
BAR_0 Bit Size	0x2	Parameter Name : BAR0_TYPE_0 BAR0 Type - 32 or 64bit	Y	Y
Allow Reprogramming of BAR_0 Mask	0x0	Parameter Name : BAR0_MASK_WRITABLE_0 When set enables dynamic changing of BAR0 Mask through DBI	Y	Y
BAR_0 Mask	0xfffff	Parameter Name : BAR0_MASK_0 BAR0 Mask ex: 64'hFFFFF = BAR size of 2ˆ20.	Y	Y
Specify Target Interface for BAR_0	0x1	Parameter Name : MEM_FUNC0_BAR0_TARGET_MAP 1 – target 1 intended destination for request matching function 0/ bar 0 0 – target 0 intended destination for request matching function 0/ bar 0 Note: This feature is not applicable for RC.	Y	Y
Enable BAR_1	0x0	Parameter Name : BAR1_ENABLED_0 BAR1 Enable	Y	Y

Parameter	Value	Description	Default?	Disabled
BAR_1 is Memory or I/O	0x0	Parameter Name : MEM1_SPACE_DECODER_0 BAR1 Memory Space Indicator When set indicates IO space	Y	Y
BAR_1 is Prefetchable	0x0	Parameter Name : PREFETCHABLE1_0 BAR1 Memory Prefetchable When set indicates BAR1 Memory BAR is a prefetchable BAR	Y	Y
BAR_1 Bit Size	0x0	Parameter Name : BAR1_TYPE_0 BAR1 Type - 32 or 64bit	Y	Y
Allow Reprogramming of BAR_1 Mask	0x0	Parameter Name : BAR1_MASK_WRITABLE_0 When set enables dynamic changing of BAR1 Mask through DBI	Y	Y
BAR_1 Mask	0xffffffff	Parameter Name : BAR1_MASK_0 BAR1 Mask ex: 64'hFFFFF = BAR size of 2ˆ20.	Y	Y
Specify Target Interface for BAR_1	0x1	Parameter Name : MEM_FUNC0_BAR1_TARGET_MAP 1 – target 1 intended destination for request matching function 0/ bar 1 0 – target 0 intended destination for request matching function 0/ bar 1 Note: This feature is not applicable for RC.	Y	Y

Function 1

Parameter	Value	Description	Default?	Disabled?
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_1 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	Y

Function 1 -> PCI Express Capability:

Function 2

Parameter	Value	Description	Default?	Disabled?
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_2 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	Y

Function 2 -> PCI Express Capability:

Function 3

Parameter	Value	Description	Default?	Disabled?
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_3 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	Y

Function 3 -> PCI Express Capability:

Function 4

Parameter	Value	Description	Default?	Disabled?

Parameter	Value	Description	Default?	Disabled?
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_4 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	Y

Function 4 -> PCI Express Capability:

Function 5

Parameter	Value	Description	Default?	Disabled?
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_5 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	Y

Function 5 -> PCI Express Capability:

Function 6

Parameter	Value	Description	Default?	Disabled?
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_6 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	Y

Function 6 -> PCI Express Capability:

Function 7

Parameter	Value	Description	Default?	Disabled?
Extended Tag Support	0x0	Parameter Name : DEFAULT_EXT_TAG_FIELD_SUPPORTED_7 Indicates the maximum supported size of the Tag field as a Requester and the ability of accepting request with 8-bit tag. Should only be set when CX_REMOTE_MAX_TAG is set to 256	Y	Y

Function 7 -> PCI Express Capability:

Filter Configuration

Parameter	Value	Description	Default?	Disabled?
FLT_Q_ADDR_WIDTH	64	Parameter Name : FLT_Q_ADDR_WIDTH number of bits for Filter field FLT_Q_ADDR	Y	Y
Allow AER (UR/CA Error) for TLPs Destined for Target 1	0x0	Parameter Name : CX_MASK_UR_CA_4_TRGT1 1 - Allow AER (UR/CA error) for TLPs destined for Trgt1 0 - Suppressed AER (UR/CA error) for TLPs destined for Trgt1	Y	
FLT Message Drop	0x1	Parameter Name : FLT_DROP_MSG Control whether or not messages are passed along to the application or consumed by the core.	Y	

Queuing & Buffer Configuration

Queue Depth Worksheet

Parameter	Value	Description	Default?	Disabled?
Enable Auto Size of Retry Buffer	0x0	Parameter Name : CX_RBUF_AUTOSIZE Switch ON / OFF automatic retry buffer sizing. When ON the retry buffer size is derived from the Maximum Payload Size, the Link Width and the core latencies. The SOTBUF Buffer size will be calucated from these same criteria. When OFF the retry buffer size must be specified by the user by entering this sizes directly in RBUF depth, and SOTBUF depth.		
MAC Tx Delay	4	Parameter Name : CX_PHY_TX_DELAY_MAC Transmitter delay (MAC) in clock cycles	Y	Y
PHY Tx Delay	5	Parameter Name : CX_PHY_TX_DELAY_PHY Transmitter delay (PHY) in clock cycles	Y	Y
MAC Rx Delay	4	Parameter Name : CX_PHY_RX_DELAY_MAC Receiver delay (MAC) in clock cycles	Y	Y
PHY Rx Delay	6	Parameter Name : CX_PHY_RX_DELAY_PHY Receiver delay (PHY) in clock cycles	Y	Y
Internal Delay / Link Partner Delay	19	Parameter Name : CX_INTERNAL_DELAY The internal processing delays for received TLPs and transmitted DLLPs. This value is used to caclculate Retry buffer and SOTBUF buffer sizes.	Y	Y

Parameter	Value	Description	Default?	Disabled?
Retry Buffer Depth	215	Parameter Name : CX_RBUF_DEPTH Number of locations in Retry Buffer RAM	Y	
Retry Buffer Width	68	Parameter Name : RBUF_WIDTH Width of Retry Buffer RAM (number of address bits)	Y	Y

Retry Buffer Configuration:

Parameter	Value	Description	Default?	Disabled?
Minimum SOT Depth	32	Parameter Name : CX_SOTBUF_DEPTH Minimum Number of RAM entries per packet. Actual sotbuf depth is adjusted to be at least 32, and will be rounded up to the next power-of-2.		
SOT Buffer Depth	32	Parameter Name : SOTBUF_DEPTH Number of locations in SOTBUF RAM	Y	Y
SOT Buffer Width	8	Parameter Name : SOTBUF_WIDTH Width of SOTBUF RAM (number of address bits)	Y	Y

SOT Buffer Configuration:

General Configuration

Parameter	Value	Description	Default?	Disabled?

Parameter	Value	Description	Default?	Disabled?
Specify Queue Mode	2	Parameter Name : CX_RADMQ_MODE There are two Queue mode supported: Multi-Q mode: Queue's are separated based into individual TLP queues. Single-Q mode: Queues that are not bypassed, will be combined into a single header queue, and a single data queue. The Posted Queue is the 'host' queue used as the Single Queue, therefore single qmode is not supported if posted queue is bypassed. Segment Buffer: (available in an upcoming release) Queues that are not bypassed are located on a single RAM but are functionally treated as separate queues.		
Inhibit RAM read enable when segment empty	1	Parameter Name : CX_RADM_ADDR_COMP Inhibits the ram's read enable when the read and write addresses are equal. Turning this option off will improve timing but may not be supported by some ram implementations. NOTE: The core only requires that the write data be written to the ram in this situation. The read data is not used and can be x's.	Y	

Parameter	Value	Description	Default?	Disabled?
Receive VC Arbitration	0x0	Parameter Name : CX_RADM_STRICT_VC_PRIORITY Arbitration between VC. If set to strict VC Priority, VC0 is lowest priority, VC7 is highest	Y	
Support Relaxed Ordering	0	Parameter Name : RELAXED_ORDER_SUPPORT Relaxed Order Support When set allows CPL types to go out of order	Y	
Enable Support for Cut-Through Mode	0	Parameter Name : CUT_THROUGH_INVOLVED	Y	
Enable Passing of ECRC Values to the Application	0	Parameter Name : ECRC_ERR_PASS_THROUGH	Y	
Enable Dynamic FC Credit Adjustment	1	Parameter Name : CX_DYNAMIC_FC_CREDIT		
Enable Dynamic Q Depth Adjustment	1	Parameter Name : CX_DYNAMIC_SEG_SIZE		
PCIe Ordering Rules Support	1	Parameter Name : CLUMP_SUPPORT PCIe Ordering Rules support This option enables support for the PCIe Ordering Rules arbitration mode. If this option is not set, PCIe Ordering Rule based Arbitration will not be available.	Y	

Segment Buffer Options:

Parameter	Value	Description	Default?	Disabled?
Posted Q Use Ordering FIFO	1	Parameter Name : CX_RADMQ_P_NB_ORDER_LIST If Posted TLP Queues are not bypassed, this parameter provides a switch to control whether the order fifo effects Posted queue operations. If the bit is set to 1, presentation of received posted TLPs is controlled by the Order FIFO. If the bit is set to 0, presentation of received posted TLPs will not be influenced by the Order FIFO.	Y	Y

Parameter	Value	Description	Default?	Disabled?
Non-Posted Q Use Ordering FIFO	1	Parameter Name : CX_RADMQ_NP_NB_ORDER_LIST If Non-Posted TLP Queues are not bypassed, this parameter provides a switch to control whether the order fifo effects Non-Posted queue operations. If the bit is set to 1, presentation of received Non-Posted TLPs is controlled by the Order FIFO. If the bit is set to 0, presentation of received Non-Posted TLPs will not be influenced by the Order FIFO.	Y	Y
Completion Q Use Ordering FIFO	0	Parameter Name : CX_RADMQ_CPL_NB_ORDER_LIST If Completion TLP Queues are not bypassed, this parameter provides a switch to control whether the order fifo effects Completion queue operations. If the bit is set to 1, presentation of received Completion TLPs is controlled by the Order FIFO. If the bit is set to 0, presentation of received Completion TLPs will not be influenced by the Order FIFO.	Y	Y

Multi Queue Options:

VC Configuration

In Single Queue and Multi-queue mode these settings are for ALL VC's

Posted Advertised Credits

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC0 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	
Hdr	59	Parameter Name : RADM_PQ_HCRD_VC0 Specifies the # of Posted Hdr Credits to Advertise.		
Data	105	Parameter Name : RADM_PQ_DCRD_VC0 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data		

Non-Posted Advertised Credits

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC0 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	
Hdr	59	Parameter Name : RADM_NPQ_HCRD_VC0 Specifies the # of Non-Posted Hdr Credits to Advertise.		

Parameter	Value	Description	Default?	Disabled?
Data	16	Parameter Name : RADM_NPQ_DCRD_VC0 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data		

Completion Advertised Credits

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC0 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.		
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC0 Specifies the # of Completion Hdr Credits to Advertise.	Y	
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC0 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	

Additional VC 0 Options

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC0 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules		
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC0 Selecting this option allow RAM depths to be specified independently from the advertised credits.		

Posted Buffer Depth

Parameter	Value	Description	Default?	Disabled?
Hdr	60	Parameter Name : RADM_PQ_HDP_VC0 Specifies the depth of the Posted Hdr Queue/RAM.	Y	
Data	211	Parameter Name : RADM_PQ_DDP_VC0 Specifies the depth of the Posted Data Queue/RAM.	Y	

Non-Posted Buffer Depth

Parameter	Value	Description	Default?	Disabled?
Hdr	60	Parameter Name : RADM_NPQ_HDP_VC0 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	
Data	33	Parameter Name : RADM_NPQ_DDP_VC0 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	

Completion Buffer Depth

Parameter	Value	Description	Default?	Disabled?
Hdr	5	Parameter Name : RADM_CPLQ_HDP_VC0 Specifies the depth of the Completion Hdr Queue/RAM.		
Data	9	Parameter Name : RADM_CPLQ_DDP_VC0 Specifies the depth of the Completion Data Queue/RAM.		

VC 1

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC1 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_PQ_HCRD_VC1 Specifies the # of Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_PQ_DCRD_VC1 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC1 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_NPQ_HCRD_VC1 Specifies the # of Non-Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DCRD_VC1 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Non-Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC1 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC1 Specifies the # of Completion Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC1 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Completion Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC1 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules	Y	Y
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC1 Selecting this option allow RAM depths to be specified independantly from the advertised credits.	Y	Y

Additional VC 1 Options:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_PQ_HDP_VC1 Specifies the depth of the Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_PQ_DDP_VC1 Specifies the depth of the Posted Data Queue/RAM.	Y	Y

Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_NPQ_HDP_VC1 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DDP_VC1 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	Y

Non-Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_CPLQ_HDP_VC1 Specifies the depth of the Completion Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DDP_VC1 Specifies the depth of the Completion Data Queue/RAM.	Y	Y

Completion Buffer Depth:

VC 2

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC2 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_PQ_HCRD_VC2 Specifies the # of Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_PQ_DCRD_VC2 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC2 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_NPQ_HCRD_VC2 Specifies the # of Non-Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DCRD_VC2 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Non-Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC2 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC2 Specifies the # of Completion Hdr Credits to Advertise.	Y	Y

Parameter	Value	Description	Default?	Disabled?
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC2 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Completion Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC2 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules	Y	Y
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC2 Selecting this option allow RAM depths to be specified independantly from the advertised credits.	Y	Y

Additional VC 2 Options:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_PQ_HDP_VC2 Specifies the depth of the Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_PQ_DDP_VC2 Specifies the depth of the Posted Data Queue/RAM.	Y	Y

Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_NPQ_HDP_VC2 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DDP_VC2 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	Y

Non-Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_CPLQ_HDP_VC2 Specifies the depth of the Completion Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DDP_VC2 Specifies the depth of the Completion Data Queue/RAM.	Y	Y

Completion Buffer Depth:

VC 3

Parameter	Value	Description	Default?	Disabled?

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC3 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_PQ_HCRD_VC3 Specifies the # of Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_PQ_DCRD_VC3 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC3 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_NPQ_HCRD_VC3 Specifies the # of Non-Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DCRD_VC3 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Non-Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC3 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC3 Specifies the # of Completion Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC3 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Completion Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC3 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules	Y	Y
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC3 Selecting this option allow RAM depths to be specified independantly from the advertised credits.	Y	Y

Additional VC 3 Options:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_PQ_HDP_VC3 Specifies the depth of the Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_PQ_DDP_VC3 Specifies the depth of the Posted Data Queue/RAM.	Y	Y

Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_NPQ_HDP_VC3 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DDP_VC3 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	Y

Non-Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_CPLQ_HDP_VC3 Specifies the depth of the Completion Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DDP_VC3 Specifies the depth of the Completion Data Queue/RAM.	Y	Y

Completion Buffer Depth:

VC 4

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC4 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_PQ_HCRD_VC4 Specifies the # of Posted Hdr Credits to Advertise.	Y	Y

Parameter	Value	Description	Default?	Disabled?
Data	0	Parameter Name : RADM_PQ_DCRD_VC4 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC4 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_NPQ_HCRD_VC4 Specifies the # of Non-Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DCRD_VC4 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Non-Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC4 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC4 Specifies the # of Completion Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC4 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Completion Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC4 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules	Y	Y
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC4 Selecting this option allow RAM depths to be specified independantly from the advertised credits.	Y	Y

Additional VC 4 Options:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_PQ_HDP_VC4 Specifies the depth of the Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_PQ_DDP_VC4 Specifies the depth of the Posted Data Queue/RAM.	Y	Y

Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_NPQ_HDP_VC4 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DDP_VC4 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	Y

Non-Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_CPLQ_HDP_VC4 Specifies the depth of the Completion Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DDP_VC4 Specifies the depth of the Completion Data Queue/RAM.	Y	Y

Completion Buffer Depth:

VC 5

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC5 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_PQ_HCRD_VC5 Specifies the # of Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_PQ_DCRD_VC5 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC5 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_NPQ_HCRD_VC5 Specifies the # of Non-Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DCRD_VC5 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Non-Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC5 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC5 Specifies the # of Completion Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC5 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Completion Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC5 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules	Y	Y
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC5 Selecting this option allow RAM depths to be specified independantly from the advertised credits.	Y	Y

Additional VC 5 Options:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_PQ_HDP_VC5 Specifies the depth of the Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_PQ_DDP_VC5 Specifies the depth of the Posted Data Queue/RAM.	Y	Y

Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_NPQ_HDP_VC5 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DDP_VC5 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	Y

Non-Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_CPLQ_HDP_VC5 Specifies the depth of the Completion Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DDP_VC5 Specifies the depth of the Completion Data Queue/RAM.	Y	Y

Completion Buffer Depth:

VC 6

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC6 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_PQ_HCRD_VC6 Specifies the # of Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_PQ_DCRD_VC6 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC6 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_NPQ_HCRD_VC6 Specifies the # of Non-Posted Hdr Credits to Advertise.	Y	Y

Parameter	Value	Description	Default?	Disabled?
Data	0	Parameter Name : RADM_NPQ_DCRD_VC6 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Non-Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC6 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC6 Specifies the # of Completion Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC6 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Completion Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC6 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules	Y	Y
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC6 Selecting this option allow RAM depths to be specified independantly from the advertised credits.	Y	Y

Additional VC 6 Options:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_PQ_HDP_VC6 Specifies the depth of the Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_PQ_DDP_VC6 Specifies the depth of the Posted Data Queue/RAM.	Y	Y

Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_NPQ_HDP_VC6 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DDP_VC6 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	Y

Non-Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_CPLQ_HDP_VC6 Specifies the depth of the Completion Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DDP_VC6 Specifies the depth of the Completion Data Queue/RAM.	Y	Y

Completion Buffer Depth:

VC 7

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_P_QMODE_VC7 Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no Posted receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: P TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: P TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_PQ_HCRD_VC7 Specifies the # of Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_PQ_DCRD_VC7 Specifies the # of Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_NP_QMODE_VC7 Non-Posted TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no NP receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: NP TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: NP TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_NPQ_HCRD_VC7 Specifies the # of Non-Posted Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DCRD_VC7 Specifies the # of Non-Posted Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Non-Posted Advertised Credits:

Parameter	Value	Description	Default?	Disabled?

Parameter	Value	Description	Default?	Disabled?
Mode	0x1	Parameter Name : RADM_CPL_QMODE_VC7 Completion TLP queue type. There are three Queue types available Bypass/Store-Forward/CutThrough. Bypass: There is no CPL receive queue in this mode, the application must be able to accept all traffic - as back-pressure is disabled in the mode. Store-Forward: CPL TLP's are stored into queue, advertisment of an available TLP is advertised only after the entire TLP is stored into the queue. Cut-Through: CPL TLP's are stored into queue and presented to the application at the same time it is being stored into the queue.	Y	Y
Hdr	0	Parameter Name : RADM_CPLQ_HCRD_VC7 Specifies the # of Completion Hdr Credits to Advertise.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DCRD_VC7 Specifies the # of Completion Data Credits to Advertise. One data credit = 128 bits of data	Y	Y

Completion Advertised Credits:

Parameter	Value	Description	Default?	Disabled?
Receive Arbitration Betweeen Types	0x1	Parameter Name : CX_RADM_ORDERING_RULES_VC7 Arbitration between transaction types (P/NP/CPL). If set to strict priority, P is higher than CPL is higher than NP Otherwise, it's set to follow PCIe spec, Table 2-23 ordering rules	Y	Y
Decouple Depth from Credit	1	Parameter Name : RADM_DEPTH_DECOUPLE_VC7 Selecting this option allow RAM depths to be specified independantly from the advertised credits.	Y	Y

Additional VC 7 Options:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_PQ_HDP_VC7 Specifies the depth of the Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_PQ_DDP_VC7 Specifies the depth of the Posted Data Queue/RAM.	Y	Y

Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_NPQ_HDP_VC7 Specifies the depth of the Non-Posted Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_NPQ_DDP_VC7 Specifies the depth of the Non-Posted Data Queue/RAM.	Y	Y

Non-Posted Buffer Depth:

Parameter	Value	Description	Default?	Disabled?
Hdr	0	Parameter Name : RADM_CPLQ_HDP_VC7 Specifies the depth of the Completion Hdr Queue/RAM.	Y	Y
Data	0	Parameter Name : RADM_CPLQ_DDP_VC7 Specifies the depth of the Completion Data Queue/RAM.	Y	Y

Completion Buffer Depth:

AXI Configuration

Parameter	Value	Description	Default?	Disabled?
AXI Enable	0	No description available.	Y	Y

Master Interface Options

Parameter	Value	Description	Default?	Disabled?
Master Interface Enable	0	Parameter Name : MASTER_POPULATED Indicates that a master interface is required	Y	Y
Enable Independent AXI Master Clock	0	No description available.	Y	Y
Master Decomposer Enable	0	Parameter Name : RADMX_DECOMPOSER_POPULATED Indicates that master interface requires a decomposer	Y	Y
Maximum Master Tags Supported	32	Parameter Name : CC_MAX_MSTR_TAG Specifies the maximum number of tags supported by the AXI Master.	Y	Y
Remote Device MAX Read Request Size	128	Parameter Name : CX_REMOTE_RD_REQ_SIZE Specifies the maximum read request size supported by the PCIe core receiver when AXI or AHB is populated AXI Master. This parameter is used to size AXI/AHB master composer memories.	Y	Y
AXI Master Address Width	32	Parameter Name : CC_MSTR_BUS_ADDR_WIDTH Specify the master address width on AXI.	Y	Y
AXI Master Data Width	32	Parameter Name : CC_MSTR_BUS_DATA_WIDTH Specify the master data width on AXI.	Y	Y
Master Page Boundary Size	13	Parameter Name : CC_MSTR_PAGE_BOUNDARY_PW Specifies the page boundary size supported by AXI Master. No packets can have an address that crosses this boundary. Packets will be split to conform to this requirement.	Y	Y

Parameter	Value	Description	Default?	Disabled?
Master Response's HEADER FIFO Queue Depth	4	Parameter Name : CC_XADMX_CLIENT0_QUEUE_HDP Indicates that bridge's master response HEADER FIFO queue size	Y	Y
Master Response's DATA FIFO Queue Depth	128	Parameter Name : CC_XADMX_CLIENT0_QUEUE_DDP Indicates that bridge's master response DATA FIFO queue size	Y	Y
Master Request's HEADER FIFO Queue Depth	4	Parameter Name : CC_RADMX_DECOMPOSER_HDRQ_DP Indicates that bridge's master request HEADER FIFO queue size	Y	Y
Master Request's DATA FIFO Queue Depth	16	Parameter Name : CC_RADMX_DECOMPOSER_DATAQ_DP Indicates that bridge's master request DATA FIFO queue size	Y	Y

Master Queue Options:

Slave Interface Options

Parameter	Value	Description	Default?	Disabled?

Parameter	Value	Description	Default?	Disabled?
Slave Interface Enable	0	Parameter Name : SLAVE_POPULATED Indicates that a slave interface is required	Y	Y
Enable Independent AXI Slave Clock	0	No description available.	Y	Y
Slave Composer Enable	0	Parameter Name : RADMX_COMPOSER_POPULATED Indicates that slave interface requires a composer	Y	Y
Maximum Slave Tags Supported	32	Parameter Name : CC_MAX_SLV_TAG Specifies the maximum number of tags supported by the AXI Slave.	Y	Y
AXI Slave Data Width	32	Parameter Name : CC_SLV_BUS_DATA_WIDTH Specify the slave data width on AXI.	Y	Y
AXI Slave Address Width	32	Parameter Name : CC_SLV_BUS_ADDR_WIDTH Specify the slave address width on AXI.	Y	Y
AXI Slave ID Width	5	Parameter Name : CC_SLV_BUS_ID_WIDTH Specify the slave ID width on AXI.	Y	Y
Enable in order services of AXI SLAVE	0	Parameter Name : SLAVE_IN_ORDER_EN Indicates that slave logic will ensure that the responses will be returned in order.	Y	

Parameter	Value	Description	Default?	Disabled?
Slave Request's HEADER FIFO Queue Depth	4	Parameter Name : CC_XADMX_CLIENT1_QUEUE_HDP Indicates that bridge's slave request HEADER FIFO queue size	Y	Y
Slave Request's DATA FIFO Queue Depth	16	Parameter Name : CC_XADMX_CLIENT1_QUEUE_DDP Indicates that bridge's slave request DATA FIFO queue size	Y	Y

Slave Queue Options:

DBI Slave Interface Options

Parameter	Value	Description	Default?	Disabled?
Slave DBI Enable	0	Parameter Name : DBI_4SLAVE_POPULATED Indicates that slave interface requires DBI	Y	Y
Enable Independent AXI DBI Slave Clock	0	No description available.	Y	Y
AXI DBI Slave Address Width	32	Parameter Name : CC_DBI_SLV_BUS_ADDR_WIDTH Specify the slave address width on AXI.	Y	Y

13.16 PCS, PHY Layers

Please reference the Synopsys' "PCI-Express 90nm PHY Data Book". This provides a description of the pins, the timing requirements, and the programmer-visible registers.

13.17 Power Management

The PCI-Express subsystem is active in only a fraction of the ICE9 chips on a processing module. To minimize power consumption, the PCI-Express subsystem must be capable of complete power-down when not in use. Support of intermediate power states is not required.

Chapter 14

I2C Interface

[Last Modified $Id: chipi2c.lyx 50693 2008-02-07 16:01:46Z wsnyder $]

14.1 Overview

The chip implements an I2C Master Controller in order to read the Serial Presence Detect (SPD) configuration of its local DIMMs using the industry standard I2C Bus.[1] This chapter provides a brief description of the I2C Master Controller, the registers provided to program it and the actions necessary to initialize and operate it.

14.2 Description

The ICE9 implementation uses the OpenCores (www.opencores.org) I2C Master Controller. The I2C core will be contained in the BBS unit with the other programmed I/O devices. The core need only generate 7-bit I2C addresses and will be operated at a frequency of 100kHz.[2] In our implementation the I2C core will be the sole I2C Bus master and should never have to arbitrate for bus mastership even though the core supports it. *Our implementation does NOT support interrupts and all mention of interrupts in the OpenCores documentation should be ignored*. See section 14.7 for descriptions of how to poll the I2C core to determine when it is no longer busy. The core specification and programmer's guide from OpenCores can be found on the WIKI at:

`http://apollo.sicortex.com/swiki/I2cInterface`

For a complete description of the I2C Bus Architecture see the Philips Semiconductors I2C Bus Specification at:

`file:///net/sicortex/system/standards/PHILIPS_I2C_spec.pdf`

14.3 Package Attributes

Package

chip_i2c_spec

14.4 Registers and Definitions

All registers in the I2C Core can be considered 8 bits wide. Although the Clock Prescale Register is internally 16 bits wide, it is read and written in two 8 bit halves and can therefore be considered as two 8-bit registers. All registers described here are implemented as per the specification on the WIKI. The addressing, however, is somewhat different. Each address is relative to the I2C Interface's base address. Register 0 starts at I2C_BASE + 0, register 1 starts at I2C_BASE+8, and so on. That is, the registers appear in the address space to be 8 bytes apart

[1] Also known as the Inter-Integrated Circuit Bus or I^2C Bus. Throughout this document it is simply referred to as the I2C Bus.

[2] Since the I2C Bus is usually transferring 1-bit of serial data on its SDA line per clock, the SCL frequency is sometimes also described in terms of a bit rate, in bits per second, scaled appropriately as either kilobits per second (kbps) or megabits per second (Mbps). Thus 100kHz = 100kbps.

even though only one byte is being transferred. For transfers within the I2C address space, the byte transferred is always the little-endian least significant byte of a 32-bit longword. *Please note that all reserved bits are read as zeros. To ensure forward compatibility, they should be written as zeros.*

14.4.1 I2C Clock Prescale Register

Description

This register is used to prescale the I2C's SCL clock line. The prescale register is 16 bits wide but must be written as two 8 bit halves, with each half at its own unique address as shown below. Due to the structure of the I2C interface, the core uses a 5*SCL clock internally. The prescale register must be programmed to this 5*SCL frequency minus 1. *You may change the value of the prescale register only when the EN bit in the control register is cleared (disabled).*

In this implementation, the I2C core derives its SCL clock from the L2 Cache clock (CCLK). With a 16 bit prescale register, this implies that the SCL clock can run at any frequency from ˜763 Hz to 50 MHz. However because I2C is an industry standard implemented by many different vendors using various processes, the I2C specification establishes standard maximum I2C clock frequencies of 100 kHz (normal), 400 kHz (fast) and 3.4 MHz (high-speed). In order to support the broadest range of devices available, this implementation should operate at the lowest standard maximum clock frequency of 100 kHz. Therefore the value for the prescale register should be chosen such that the operating CCLK frequency is divided down to 100 kHz.

The formula for calculating the prescale value is:

$$prescale = \frac{cclk}{5 * scl} - 1$$

Substituting our known frequency values for *cclk* and *scl* yields:

$$prescale = \frac{250,000,000}{5 * 100,000} - 1 = 499 = 1F3(hex)$$

The two halves used to read and write the prescale register are as follows:

Register

R_I2cPrerLo

Address

0xE_A800_0000

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved
7:0	prerlo	RW	0xFF		Low byte of I2C clock prescale register. Change only when EN bit of I2C Control Register is '0'.

Register

R_I2cPrerHi

Address

0xE_A800_0008

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved
7:0	prerhi	RW	0xFF		High byte of I2C clock prescale register. Change only when EN bit of I2C Control Register is '0'.

14.4.2 I2C Control Register

Description

The Control Register enables I2C operation. The core responds to new commands only when the EN bit is set and after pending commands are finished. Clear the EN bit only when no transfer is in progress, i.e. after a STOP command, or when the command register has the STO bit set. If halted during a transfer, the core can hang the I2C Bus.

Register

R_I2cCtl

Address

0xE_A800_0010

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved
7	en	RWS	0		Enable I2C unit. When 1, the I2C widget is enabled.
6:0			0		Reserved

14.4.3 I2C Data Register

Description

On a write, contains next byte to send onto the I2C Bus from the master core. The byte can be either data or the 7-bit I2C slave address along with the read/write command. On a read, contains the last byte received from the I2C Bus.

Register

R_I2cData

Address

0xE_A800_0018

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved
7:0	rxData	R	X		Last byte received from the I2C bus. Overlaps allowed
7:0	txData	WS	0		Next byte to transmit on the I2C bus. Overlaps allowed.

Bit	Mnemonic	Access	Reset	Type	Definition
7:1	txAddr	W	0		For slave address transfers these bits represent the 7-bit I2C address. Overlaps allowed.
0	txRW	W	0		For slave address transfers this bit represents the I2C R/W bit. '1' = reading from slave '0' = writing to slave Overlaps allowed.

14.4.4 I2C Command and Status Register

Description

Controls the operation of the I2C Master core on write and reports its status on read. See the core specification on the WIKI and the transfer sequences described in this document for a more detailed description on how to use the bits in this register. *Note that the STA, STO, RD, and WR bits are cleared automatically. These bits are always read as zeros.*

Register

R_I2cCmdSts

Attributes

-writeonemixed

Address

0xE_A800_0020

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved
7	sta	WS	0		Generate start or repeated-start condition. Overlaps allowed
6	sto	WS	0		Generate stop condition. Overlaps allowed
5	rd	WS	0		Read data from slave. Overlaps allowed
4	wr	WS	0		Write data to slave. Overlaps allowed
3	ack	W1C	0		When acting as a receiver, send ACK (ACK='0') or NACK (ACK='1'). Overlaps allowed
2:0		W	0		Reserved. Write as zero. Overlaps allowed
7	rxack	R	0		Received acknowledge from slave. This flag represents acknowledge from the addressed slave. '1' = No acknowledge received '0' = Acknowledge received Overlaps allowed

Bit	Mnemonic	Access	Reset	Type	Definition
6	busy	R	0		I2C bus busy. Use this flag to determine when a forced stop operation is complete. A forced stop occurs when *only* the STO bit in the command register is set. A return value of '0' indicates the operation has completed. '1' after START signal detected. '0' after STOP signal detected. Overlaps allowed
5:2		R	0		Reserved Overlaps allowed
1	tip	R	0		Transfer in progress. Use this flag to determine when a transfer is complete after either the RD or WR bit has been set in the Command Register. '1' when transferring data '0' when transfer is complete Overlaps allowed
0		R	0		Reserved Overlaps allowed

14.4.5 I2C Core Reset Register

Description

Provides a software controllable reset to the I2C core. This register is not actually part of the I2C Core logic. It is implemented in the CSI widget of the PMI and is used to drive the synchronous software-based reset to the I2C core. A write of any value to this register will assert the synchronous reset to the I2C Core for one CCLK cycle.

Register

R_I2cReset

Address

0xE_A800_0028

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	reset	WS	0		I2C Core Reset A write of any value will reset the I2C core.

14.5 Reset

The I2C Core can be reset under both hardware and software control. The hardware reset is provided at power-on and under Module Service Processor control via the I2C Reset Control Bit in the Reset Control Register portion of the SysChain implemented in the LBS. The hardware reset asserts asynchronously and releases synchronous to CCLK. The ARST_LVL core parameter described in the OpenCore spec is left unchanged so that the core supports an active low asynchronous hardware reset. The software reset is provided by the *R_I2cReset* register. Writing any value to this register will reset the I2C Core synchronous to CCLK by asserting reset for one CCLK cycle.

14.6 Initialization

For the ICE9 implementation the I2C Core exits reset synchronous to CCLK. During reset the following actions occur:

- The Prescale Register is set to 0xFFFF the slowest I2C clock speed available.
- The EN bit in the Control Register is cleared, disabling the core.
- The Transmit and Receive Data Registers are both cleared.
- All bits in the Command Register are cleared.
- The I2C Master Controller is placed into the idle state.
- The I2C bus drivers are disabled, allowing the SCL and SDA wires to rise to a logic level of '1'.

After reset, software should perform the following operations in the order listed to prepare the core for normal operation:

1. Set the Prescale Registers to the correct value for a 100kHz I2C SCL frequency. You may write the halves in any order, but it is probably easiest to write the MSB first and the LSB last.
2. Set the EN bit in the Control Register.

14.7 Transfer Sequences

14.7.1 Example 1: Byte Writes

Write to a slave memory device at I2C address 0x51, 1 byte of data (0xAC) to location 128 (0x80). To write multiple bytes; simply repeat commands 9 to 12 below, but DO NOT set the STO bit in the Command Register until sending the last byte. *Note: Typically a slave memory device will wrap back to its first location when writing past the last location of the device.* ***Extra caution should be observed when writing to a DIMM SPD Serial-EEPROM because of this behavoir. Also, SPD devices typically support multi-byte writes only up to ablock size of 16 bytes. They may wrap around to the start address after 16 bytes.***[3]

I2C-Sequence:

1. Generate a START command.
2. Send the slave device address + the write bit.
3. Wait for an acknowledge from the slave.
4. Write the address to be written.
5. Wait for an acknowledge from the slave.
6. Write the data to be written.
7. Wait for an acknowledge from the slave.
8. Generate a STOP command.

Commands:

1. Write 0xA2 (address 0x51 left shifted 1 bit to accomodate r/w bit + write bit of '0') to the Transmit Data Register.
2. Set the STA and WR bits in the Command Register.

[3]Some Serial-EEPROM devices offer an I2C programmable write-protect feature. This feature prevents the writing of any data into the device without first writing a special data pattern to a specific location to unlock the device. Writing a different special data pattern or a different specific location will re-lock the device when finished.

3. Poll TIP flag in the Status Register until it is negated.
4. Read RxACK bit from the Status Register, should be '0'.
5. Write 0x80 (address to be written, location 128 decimal) to the Transmit Data Register.
6. Set WR bit in the Command Register.
7. Poll TIP flag in the Status Register until it is negated.
8. Read RxACK bit from Status Register, should be '0'.
9. Write 0xAC (the data to be written) to the Transmit Data Register.
10. Set STO and WR bits in the Command Register.
11. Poll TIP flag in the Status Register until it is negated.
12. Read RxACK bit from the Status Register, should be '0'.

14.7.2 Example 2: Byte Reads

Read from a slave memory device at I2C address 0x51, one byte of data at location 128 (0x80). To read multiple bytes, simply repeat commands 13 to 15 below for each byte to be read, but DO NOT set the ACK and STO bits in the Command Register until reading the last byte. *Note: Typically a slave memory device will wrap back to its first location when reading past the last location of the device.*

I2C-Sequence:

1. Generate a START command.
2. Write the slave address + write bit.
3. Receive acknowledge from the slave.
4. Write the memory address to the slave.
5. Receive acknowledge from the slave.
6. Generate a repeated START command.
7. Write the slave address + read bit.
8. Receive acknowledge from the slave.
9. Read a byte from the slave.
10. Write no acknowledge (NACK) to slave, indicating end of transfer.
11. Generate stop signal.

Commands:

1. Write 0xA2 (address 0x51 left shifted 1 bit to accomodate r/w bit + write bit of '0') to the Transmit Data Register.
2. Set the STA and WR bits in the Command Register.
3. Poll TIP flag in the Status Register until it is negated.
4. Read RxACK bit from the Status Register, should be '0'.
5. Write 0x80 (the memory location to be read) to the Transmit Data Register.
6. Set the WR bit in the Command Register.
7. Poll TIP flag in the Status Register until it is negated.

8. Read RxACK bit from the Status Register, should be '0'.
9. Write 0xA3 (address 0x51 left shifted 1 bit to accomodate r/w bit + read bit of '1') to the Transmit Data Register.
10. Set the STA and WR bits in the Command Register.
11. Poll TIP flag in the Status Register until it is negated.
12. Read RxACK bit from the Status Register, should be '0'.
13. Set the RD bit, the ACK bit to '1' (NACK), and the STO bit in the Command Register.
14. Poll TIP flag in the Status Register until it is negated.
15. Read the byte in the Receive Data Register that was transferred over I2C from the slave memory.

14.7.3 Example 3: Unacknowledged Transfer

In this example, no slave acknowledges the address and the master must free the I2C bus with a stop. Assume that the intended slave at I2C address 0x10 fails to acknowledge its address. In this case it is necessary to generate a stop independent of a read or write transaction. To determine when the issued stop operation has completed, it is necessary to poll the BUSY bit in the Status Register in place of the TIP bit. The TIP bit does not change when only a STOP has been issued from the Command Register.

I2C-Sequence:

1. Generate a START command.
2. Send a write to an unused slave address.
3. Receive a no-acknowledge.
4. Abort the operation by generating a stop signal.

Commands:

1. Write 0x20 (address 0x10 left shifted 1 bit to accomodate r/w bit + write bit of '0') to the Transmit Register.
2. Set the STA and WR bits in the Command Register.
3. Poll TIP flag in the Status Register until it is negated.
4. Read RxACK bit from the Status Register, should be '0' but we obtain a '1' (no ack).
5. Set the STO bit in the Command Register to force a stop.
6. Poll the BUSY flag in the Status Register until it is set to '0'.

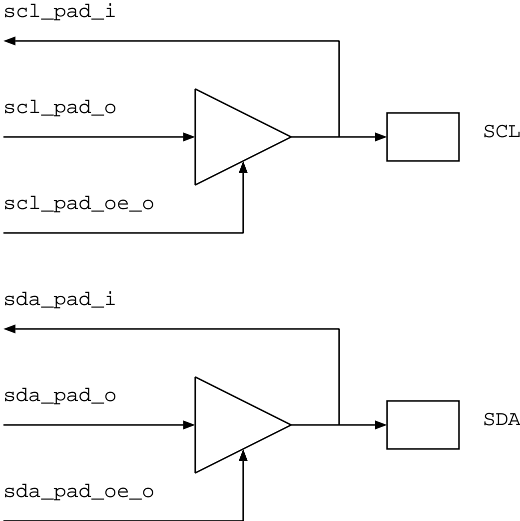
It should be noted that unacknowledged transfers can also occur on data transfers between master and slave, not just on an address as in this example. In either case, the master must abort the operation and free the I2C bus by issuing a stop. *In general, when commanding only a stop condition, the BUSY bit should be polled in place of the TIP bit to determine when the master has completed the operation.*

14.8 External Connections

The I2C interface uses a bi-directional serial data line (SDA) and a bi-directional serial clock line (SCL) for data transfers. All devices connected to these two signals must have open drain or open collector outputs. Both lines must be pulled-up to Vdd or Vcc by external resistors.

In the ICE9 implementation, the I2C core assumes open drain tri-state buffers for SDA and SCL will be added at a higher hierarchial level. Internally it uses two uni-directional signals and an output enable for each of SDA and SCL. Connections between the core and pins should be made according to the following figure:

Figure 14.1: External Connections



Chapter 15

UART

[Last Modified $Id: chipuart.lyx 50693 2008-02-07 16:01:46Z wsnyder $]

15.1 Overview

The chip implements a standard UART to support kernel debugging from a serial console line. This chapter provides a brief description of the UART, the registers provided to program the device and the actions necessary to initialize and operate the device.

15.2 Differences, Bugs, and Enhancements

15.2.1 Product and Chip Pass Differences

1. FIX NEED IMPL: TWC9A removes the UART flow control signals. They were never used on the ICE9 modules.

15.3 Description

The ICE9 implementation uses the Open Cores (www.opencores.org) 16550 UART core. This core supports the EIA RS232 serial line protocol and is Wishbone Bus compliant. For this application it has been modified to operate strictly in 8-bit mode and does not support the special debug features that were in the original core.[1] It is nearly identical in operation to the industry standard National Semiconductor 16550A with the main exceptions being that only the FIFO mode is supported and the scratch register is not implemented. For a full description, see the Open Cores specification on the WIKI at:

`http://apollo.sicortex.com/swiki/UartInterface`

The UART core will be contained in the BBS unit with the other programmed I/O devices. The UART may interrupt any of the six processors on the ICE9 node. The UART TX/RX data signals and RTS/CTS hardware flow controls are brought out to pins on the chip that may be wired to a header on the board after level conversion as well as to an external multiplexer on the Module Service Processor. This allows for both local and remote serial console access to the chip.

15.4 Package Attributes

Package

chip_uart_spec

[1] Or were intended to be in the original core. The most recent version from Open Cores that was available to us when we started had several bugs in this area. We finessed the problem by not implementing these unneeded features.

15.5 Registers and Definitions

All registers in the UART are 8 bits wide and are fully described in the UART spec on the WIKI (*see above*). All registers described here are implemented as per the specification on the WIKI. The addressing, however, is somewhat different. Each address is relative to the UART base address. Register 0 starts at UART_BASE + 0, register 1 starts at UART_BASE + 8, and so on. That is, the registers appear in the address space to be 8 bytes apart even though only one byte is being transferred. For transfers within the UART address space, the byte transferred is always the least significant byte of a little-endian 64-bit word. The UART_BASE is simply the first address used. Table 15.1 lists all of the registers implemented in the UART.

Table 15.1: UART Register List

Name	Offset	Width	Access	Description
Receiver Buffer	0	8	R	Receiver FIFO output.
Transmitter Holding Register (THR)	0	8	W	Transmit FIFO input.
Interrupt Enable	1	8	RW	Enable/Mask Interrupts generated by the UART
Interrupt Identification	2	8	R	Get interrupt information
FIFO Control	2	8	W	Control FIFO options
Line Control Register	3	8	RW	Control connection.
Modem Control	4	8	W	Modem control signals *(unused)*
Line Status	5	8	R	Status Information
Modem Status	6	8	R	Modem status *(unused)*
Divisor Latch Byte 1 (LSB)	0	8	RW	The LSB of the divisor latch.
Divisor Latch Byte 2 (MSB)	1	8	RW	The MSB of the divisor latch.

15.5.1 Baud Rate Generation using the Clock Divisor Latch

The Divisor Latch can be accessed by setting the 7^{th}bit of LCR to '1'. This bit should be set back to '0' after setting the Divisor Latch in order to restore access to the other registers that occupy the same addresses. The two bytes of the Divisor Latch form one 16-bit register, which is internally accessed as a single number. Therefore to insure normal operation, both bytes of the register should always be set. The Divisor Latch is set to the default value of 0 on reset, which disables all serial I/O operations in order to ensure explicit setup of the register by software. The value in the Divisor Latch is used to determine the baud rate of the serial I/O lines as a function of the input clock. The value set should be equal to (system clock speed) / (16 x desired baud rate). *The internal counter starts to work when the LSB of the Divisor Latch is written, so when setting the Divisor Latch, write the MSB first and the LSB last.*

In this implementation the input clock is the Level 2 Cache Clock (CCLK). The formula for computing the contents of the Divisor Latch (DIVL) based on the baud rate is:

$$divl = \frac{cclk}{(16 \times baudrate)}$$

Given a CCLK of 250MHz and a baud rate of 9600, the DIVL must be:

$$divl = \frac{250,000,000}{16 \times 9600} = 1,627.604 \rightarrow 1,628 = 65C_{hex}$$

Table 15.3 provides various DIVL settings for standard RS232 baud rates using CCLK values of 200, 225, 250 and 275 MHz. *Note: The hexadecimal values shown reflect the DIVL values rounded to the nearest integer value.*

Table 15.3: Divisor Latch Values for Common Baud Rates

Baud Rate	DIVL @ 200MHz CCLK		DIVL @ 225MHz CCLK		DIVL @ 250MHz CCLK		DIVL @ 275MHz CCLK	
300	41,666.67	$A2C3_{hex}$	46875	$B71B_{hex}$	52,083.33	$CB73_{hex}$	57291.67	$DFCC_{hex}$
600	20,833.33	5161_{hex}	23437.5	$5B8E_{hex}$	26,041.67	$65BA_{hex}$	28645.83	$6FE5_{hex}$
1200	10,416.67	$28B1_{hex}$	11718.5	$2DC7_{hex}$	13,020.83	$32DD_{hex}$	14322.92	$37F3_{hex}$
2400	5,208.33	1458_{hex}	5859.38	$16E3_{hex}$	6,510.42	$196E_{hex}$	7161.46	$1BFA_{hex}$
4800	2,604.17	$A2C_{hex}$	2929.69	$B72_{hex}$	3,255.21	$CB7_{hex}$	3580.73	DFD_{hex}
9600	1,302.08	516_{hex}	1464.84	$5B9_{hex}$	1,627.6	$65C_{hex}$	1790.36	$6FE_{hex}$
19200	651.04	$28B_{hex}$	732.42	$2DC_{hex}$	813.8	$32E_{hex}$	895.18	$37F_{hex}$
28800	434.03	$1B2_{hex}$	488.28	$1E8_{hex}$	542.53	$21F_{hex}$	596.79	255_{hex}
38400	325.52	146_{hex}	366.21	$16E_{hex}$	406.9	197_{hex}	447.59	$1C0_{hex}$
57600	217.01	$D9_{hex}$	244.14	$F4_{hex}$	271.27	$10F_{hex}$	298.39	$12A_{hex}$
115200	108.51	$6D_{hex}$	122.07	$7A_{hex}$	135.63	88_{hex}	149.2	95_{hex}

Since the protocol is asynchronous and the sampling of the bits is conducted during the middle of the bit time, it is highly immune to small differences in the clocks of the sending and receiving sides. However, no such assumption should be made when calculating the Divisor Latch values; these should be as precise as possible.

A word about the round-off errors for DIVL in the baud rate table above. The checked references indicate that it is sufficient to maintain a baud rate clock to an accuracy of 3% (or better) of the bit time.[2] To account for possible bit rate errors at both ends of the connection a 1% tolerance figure is used. For the worst case scenario of 115,200 bps the ideal bit time is $8.681uS$. 1% of the ideal bit time is $\pm 86.8nS$; therefore any error must fall within this constraint. With a rounded DIVL setting of 109, the baud rate for a worst case CCLK of 200MHz is 114,678.899 with a bit time of $8.720uS$. The error is $8.720 - 8.681 = .039uS = 39nS$ which is well within the 1% constraint.

In general; the faster the source clock, the less the susceptibility to bit rate errors due to divisor latch rounding. Even though higher baud rates have less tolerance for bit rate errors, in this implementation even the fastest RS232 baud rate is orders of magnitude slower than the source clock.

15.5.2 RX/TX Data and Divisor Latch LSB

Description

When read this register contains the output from the UART Receive FIFO. When written this register loads the input to the UART Transmit FIFO. When the 7^{th}bit of the Line Control Register is set to '1' this register contains the least significant byte of the 16-bit clock divisor latch.

Register

R_UartData

Attributes

-noregtestcpu_reset -kernel

Address

0xE_B800_0000

[2] *Determining Clock Accuracy Requirements for UART Communication* DALLAS/Maxim Application Note AN2141, see the file at /net/sicortex/system/papers/UartClockAccuracy.pdf, the TIA/EIA-232-F Standard (http://global.ihs.com), and http://www.seetron.com/ser_an1.htm, etc.

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved.
7:0	rxBuf	RS	X		*Receiver Buffer.* Output from the UART Receiver FIFO. Overlaps allowed.
7:0	txReg	WS	0		*Transmitter Holding Register.* Input to the UART Transmit FIFO. Overlaps allowed.
7:0	divl1	RWS	0		*Divisor Latch LSB.* When LCR<7>='1' this field contains the least significant byte of the 16-bit divisor latch. Overlaps allowed.

15.5.3 Interrupt Enable Register (IER) and Divisor Latch MSB

Description

The IER enables the various interrupts provided by the UART. When the 7^{th}bit of the Line Control Register is set to '1' this register contains the most significant byte of the 16-bit clock divisor latch.

Register

R_UartIntrEnb

Attributes

-kernel

Address

0xE_B800_0008

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved.
7:4					Reserved. Overlaps allowed.
3	ms	RW	0		*Enable Modem Status Interrupt.* Overlaps allowed.
2	rls	RW	0		*Enable Receiver Line Status Interrupt.* Overlaps allowed.
1	thre	RW	0		*Enable Transmitter Holding Register Empty Interrupt.* Overlaps allowed.
0	rda	RW	0		*Enable Received Data Available Interrupt.* Overlaps allowed.
7:0	divl2	RWS	0		*Divisor Latch MSB.* When LCR<7>='1' this field contains the most significant byte of the 16-bit divisor latch. Overlaps allowed.

15.5.4 Interrupt Identification Register (IIR) and FIFO Control Register (FCR)

Description

The IIR enables the programmer to retrieve the current highest priority pending interrupt. Bit 0 indicates that an interrupt is pending when it's logic '0'. When it's '1' no interrupt is pending. The FCR allows selection of the FIFO trigger level (the number of bytes in the FIFO required to enable the Received Data Available interrupt). In addition, the FIFOs can be cleared using this register. *In this implementation the maximum FIFO depth is 16 bytes for both transmit and receive FIFOs.*

Table 15.8 lists the interrupts indicated by the *intrId* field along with their relative priority, source and reset control.

Register

R_UartIntrIdFifoCtrl

Attributes

-kernel -writeonemixed

Address

0xE_B800_0010

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved. Overlaps Allowed.
7:6		R	0x3		Reserved. Overlaps Allowed.
5:4		R	0x0		Reserved. Overlaps Allowed.
3:1	intrId	R	0x0		*Interrupt Id.* (See Table 15.8 below) Overlaps Allowed.
0	intrPend	R	1		*Interrupt Pending* (active low) ’0’ - Interrupt pending. ’1’ - Interrupt not pending. Overlaps Allowed.
7:6	rxFifoTrigLvl	W	0x3		*Receive FIFO Trigger Level.* Define the Receive FIFO Interrupt trigger level. ’0x0’ - 1 byte ’0x1’ - 4 bytes ’0x2 - 8 bytes ’0x3’ - 14 bytes Overlaps Allowed.
5:3		W	0x0		Reserved. Overlaps Allowed.
2	txReset	W1C	0		*Transmit FIFO Reset.* Writing a ’1’ to this bit clears the Transmitter FIFO and resets its logic. The shift register is not cleared, i.e. transmitting of the current character continues. Overlaps Allowed.
1	rxReset	W1C	0		*Receive FIFO Reset.* Writing a ’1’ to this bit clears the Receiver FIFO and resets its logic. It does not clear the shift register, i.e. receiving of the current character continues. Overlaps Allowed.
0		W	0		Reserved. Overlaps Allowed.

Table 15.8: Interrupt ID Field Definitions

Bit 3	Bit 2	Bit 1	Priority	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	1	1	1	Receiver Line Status	Parity, Overrun or Framing errors or Break Interrupt.	Reading the Line Status Register.
0	1	0	2	Receiver Data Available	FIFO trigger level reached.	FIFO drops below trigger level.
1	1	0	3	Timeout Indication	There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 character times. *Should not occur under normal operation.*	Reading from the FIFO Receiver Data Register.
0	0	1	4	Transmitter Holding Register Empty	Transmitter Data Register is empty.	Writing to the Transmitter Data Register or reading the IIR.
0	0	0	5	Modem Status	CTS, DSR, RI or DCD. *Only CTS should trigger this interrupt under normal operation.*	Reading the Modem Status Register.

15.5.5 Line Control Register (LCR)

Description

The LCR allows the specification of the format of the asynchronous data communication used. A bit in the register also allows access to the Divisor Latches, which define the baud rate. Reading from the register is allowed to check the current settings of the communication.

Register

R_UartLineCtrl

Attributes

-kernel

Address

0xE_B800_0018

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved.
7	divl	RWS	0		*Divisor Latch Access Bit.* '0' - The normal registers are accessed. '1' - The divisor latches can be accessed.
6	breakCtrl	RW	0		*Break Control Bit.* '0' - The break state is disabled. '1' - The serial out is forced into logic '0' (break state). *Always leave at the reset value.*

Bit	Mnemonic	Access	Reset	Type	Definition
5	stickParity	RW	0		*Stick Parity Control Bit.* '0' - Stick Parity disabled. '1' - If bits 3 and 4 are logic '1', the parity bit is transmitted and checked as logic '0'. If bit 3 is '1' and bit 4 is '0' then the parity bit is transmitted and checked as '1'. *Always leave at the reset value.*
4	evenParity	RW	0		*Even Parity Select.* '0' - Odd number of '1's are transmitted and checked in each word (data and parity combined). In other words, if the data has an even number of '1's in it, then the parity bit is '1'. '1' - Even number of '1's are transmitted in each word. *Always leave at the reset value.*
3	parityEnb	RW	0		*Parity Enable.* '0' - No parity. '1' - Parity bit is generated on each outgoing character and is checked on each incoming one. *Always leave at the reset value.*
2	stopBits	RW	0		*Stop bits.* Specify the number of generated stop bits. '0' - 1 stop bit. '1' - 1.5 stop bits when 5-bit character length selected and 2 bits otherwise. Note: The receiver always checks the first stop bit only. *Always leave at the reset value.*
1:0	bitsPerChar	RW	0x3		*Bits per character.* Select number of bits in each character. '0x0' - 5 bits '0x1' - 6 bits '0x2' - 7 bits '0x3' - 8 bits *Always leave at the reset value.*

15.5.6 Modem Control Register (MCR)

Description

The MCR allows transferring control signals to a modem connected to the UART.

Register

R_UartModemCtrl

Attributes

-kernel

Address

0xE_B800_0020

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:5					Reserved.

Bit	Mnemonic	Access	Reset	Type	Definition
4	loopback	W	0		*Loopback Mode.* '0' - Normal operation. '1' - Loopback mode. When in loopback mode, the Serial Output Signal (STX_PAD_O) is set to logic '1'. The signal of the transmitter shift register is internally connected to the input of the receiver shift register. The following connections are made: DTR -> DSR RTS -> CTS Out1 -> RI Out2 -> DCD *Always leave at the reset value.*
3	out2	W	0		*Out2.* In loopback mode, connected to Data Carrier Detect (DCD) input. *Always leave at the reset value.*
2	out1	W	0		*Out1.* In loopback mode, connected to Ring Indicator (RI) signal input. *Always leave at the reset value.*
1	rts	WS	0		*Request To Send.* (RTS) Signal Control. '0' - RTS is '1' '1' - RTS is '0'
0	dtr	W	0		*Data Terminal Ready.* (DTR) Signal Control. '0' - DTR is '1' '1' - DTR is '0' *Unused in this implementation.*

15.5.7 Line Status Register (LSR)

Description

The LSR provides the operational line status for the UART. The line status consists of transmitter line and FIFO status and receiver FIFO error, break and ready indicators.

Register

R_UartLineStatus

Attributes

-kernel

Address

0xE_B800_0028

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved.
7	ei	R	0		*Receive FIFO Error.* '1' - At least one parity error, framing error, overrun error or break indications have been received and are inside the FIFO. The bit is cleared upon reading from the register. '0' - Otherwise.

Bit	Mnemonic	Access	Reset	Type	Definition
6	te	R	1		*Transmitter Empty.* '1' - Both the transmitter FIFO and transmitter shift register are empty. The bit is cleared when data is being written to the transmitter FIFO. '0' - Otherwise.
5	tfe	R	1		*Transmit FIFO Empty.* '1' - The transmitter FIFO is empty. Generates Transmitter Holding Register Empty interrupt. The bit is cleared when data is being written to the transmitter FIFO. '0' - Otherwise.
4	bi	R	0		*Break Interrupt (BI) Indicator.* '1' - A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit). In that case, one zero character enters the FIFO and the UART waits for a valid start bit to receive next character. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No break condition in the current character.
3	fe	R	0		*Framing Error (FE) Indicator.* '1' - The received character at the top of the FIFO did not have a valid stop bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No framing error in the current character.
2	pe	R	0		*Parity Error (PE) Indicator.* '1' - The character that is currently at the top of the FIFO has been received with parity error. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No parity error in the current character.
1	oe	R	0		*Overrun Error (OE) Indicator.* '1' - If the Receive FIFO is full and another character has been received in the receiver shift register. If another character is starting to arrive, it will overwrite the data in the shift register but the FIFO will remain intact. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' - No overrun state.
0	dr	R	0		*Data Ready (DR) Indicator.* '1' - At least one character has been received and is in the Receive FIFO. '0' - No characters in the Receive FIFO.

15.5.8 Modem Status Register (MSR)

Description

The MSR displays the current state of the modem control lines.

Register

R_UartModemStatus

Attributes

-kernel

Address

0xE_B800_0030

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:8					Reserved.
7	cdcd	R	1		*DCD Complement Input. Always '1'.* Or equal to Out2 in loopback mode.
6	cri	R	1		*RI Complement Input. Always '1'.* Or equal to Out1 in loopback mode.
5	cdsr	R	1		*DSR Complement Input. Always '1'.* Or equals DTR in loopback mode.
4	ccts	R	0		*CSR Complement Input.* Or equals RTS in loopback mode.
3	ddcd	R	0		*Delta Data Carrier Detect. Always '0'.* '1' - The DCD line has changed its state. '0' - Otherwise.
2	teri	R	0		*Trailing Edge of Ring Indicator. Always '0'.* '1' - The ring indicator has changed state from low to high. '0' - Otherwise.
1	ddsr	R	0		*Delta Data Set Ready. Always '0'.* '1' - If the DSR line has changed its state. '0' - Otherwise.
0	dcts	R	0		*Delta Clear To Send.* '1' - The CTS line has changed its state. '0' - Otherwise.

15.5.9 UART Enable Register

Description

The UART Enable Register allows software to observe the UART I/O Enable condition. This register is not part of the UART core but is a read-only I/O space register implemented in the Wishbone Interface (WBI) widget of the PMI. It is documented here because of its close affinity with UART operation.

Register

R_UartEnable

Attributes

-kernel

Address

0xE_B800_0040

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition

Bit	Mnemonic	Access	Reset	Type	Definition
31:1					Reserved
0	ioenb	R	0		*UART IO Enabled* '0' - If the UART I/O is not enabled at the chip pins. '1' - If the UART I/O is enabled at the chip pins. *Settable only via the SysChain from the Module Service Processor.*

15.5.10 UART Reset Register

Description

The UART Reset Register allows software to reset the UART core. This register is not part of the UART core but is a write-only I/O space register implemented in the Wishbone Interface (WBI) widget of the PMI. A write of any value to this register will perform a reset of the UART. It is documented here because of its close affinity with UART operation.

Register

R_UartReset

Address

0xE_B800_0048

Definitions

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	reset	WS	0		*UART Reset.* A write of any value resets the UART.

15.6 Reset

The UART Core can be reset under both hardware and software control. The hardware reset is provided at power-on and under Module Service Processor control via the *UART Reset Bit* in the SysChain's *Reset Control Register*. The software reset is provided by the *R_UartReset* register. Writing any value to this register will reset the UART. Upon either reset, all UART registers revert to their reset default values and it is up to software to write them with useful values afterwards.

15.7 Initialization

In the ICE9 implementation, the UART core exits reset synchronous to CCLK. During reset the core performs the following tasks:

- The receiver and transmitter FIFOs are cleared.
- The receiver and transmitter shift registers are cleared.
- The Divisor Latch register is set to 0.
- The Line Control Register is set to 0.
- All interrupts are disabled in the Interrupt Enable Register.

After reset, perform the following initializations in the order listed for normal UART operation:

1. Set the Line Control Register to the desired line control parameters. Set bit 7 to '1' to allow access to the Divisor Latches.

2. Set the Divisor Latches, MSB first, LSB last.

3. Set bit 7 of LCR to '0' to disable access to the Divisor Latches. At this time the transmission engine starts working and data can be sent and received.

4. Set the FIFO trigger level. Generally, higher trigger level values produce fewer interrupts, so setting it to 14 bytes is recommended if the system responds fast enough.

5. Enable desired interrupts by setting the appropriate bits in the Interrupt Enable Register.

15.8 Interrupts

The UART core can send an interrupt to the processors via the ICE9 interrupt logic. See the Processor Segments chapter in this specification for a complete description of how the processors handle this interrupt.

To generate a UART interrupt on reception of data; first set the encoding for the *Receive FIFO Trigger Level (rxFifoTrigLvl)* in the *FIFO Control Register (R_UartIntrIdFifoCtrl)* to the number of bytes (1, 4, 8, or 14) to be buffered in the receive FIFO before an interrupt is sent; then set the *Enable Receiver Data Available Interrupt (rdi)* bit in the *Interrupt Enable Register (R_UartIntrEnb)*. To generate a UART interrupt when sending data, set the *Enable Transmitter Holding Register Empty Interrupt (thre)* bit. To enable interrupts whenever TxCTS_L changes, set the *Enable Modem Status Interrupt (ms)* bit.

When handling a UART interrupt, the interrupt handler should examine the *Interrupt Id (intrId)* bits in the *Interrupt Identification Register (R_UartIntrIdFifoCtrl)* to determine the cause of the interrupt. See Table 15.8 for a complete description of the *Interrupt Id* bits.

15.9 External Connections

15.9.1 Module Service Processor Enabled I/O

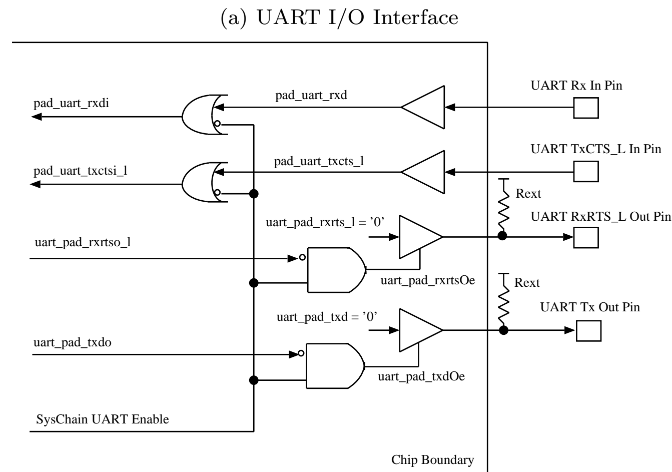
In the ICE9 implementation, the UART TX and RX data lines and hardware flow control signals are brought to pins off-chip. All off-chip UART signals are enabled by the Module Service Processor (MSP) via a bit in a shadow latch on the SysChain in the LBS unit. Figure 15.1 below is a schematic that shows how the UART I/O pad on the chip is configured.

If the UART is left disabled then the UART RX line and TxCTS_L output flow control is driven internally by the chip to a logic '1', causing the UART core to only see STOP bits with output flow control off, ignoring anything that the MSP may be writing to the line. In addition, the UART TX line and RxRTS_L input flow control are also disabled, allowing another ICE9 chip to drive the line. This is accomplished by using an open-drain driver with a hardwired input of logic '0' and an external pull-up on the Tx and RxRTS_L output pins. Whenever the SysChain UART Enable is asserted, the UART core's outputs control the enables, allowing the driver to toggle between logic levels. Otherwise the driver is left disabled and external weak pull-up resistors (Rext) is used to hold the lines at the logic '1' state, effectively driving STOP bits to the MSP with input flow control off unless another ICE9 chip is driving the wire.

This greatly simplifies the UART interconnect between the ICE9 chips and allows the MSP to control which ICE9 UART port is active.

Figure 15.1: UART External Connections

(a) UART I/O Interface



15.9.2 RS232 Line Voltage Conversion

Because the ICE9 supports I/O voltages of only 0 and +2.5 Volts on the UART pins, an external RS-232 line converter chip should be used to match voltage and logic levels to the RS-232 standard if that is desired.

Chapter 16

Addressing

[Last modified: $Id: chipaddr.lyx 43441 2007-08-17 17:38:27Z wsnyder $]

16.1 Overview

This chapter discusses the global address map. The ICE-9 physical address is 36 bits, split into half cached and half uncached IO space. This allows a maximum of 32GB of main memory.

16.2 Differences, Bugs, and Enhancements

16.2.1 Product and Chip Pass Differences

1. TWC9A adds some values to the AddrBusStop enumeration to support the additional cores, bug3377 .

16.3 Physical Address Regions

The 36-bit CPU physical address is split into the following major regions.

Start Address	End Address	Size	Access	Description
0x0_0000_0000	0x7_FFFF_FFFF	32GB	Any	Main memory - Cachable. There are some magic regions in this space, including use of the last 4GB for boot; see the Definitions.
0x8_0000_0000	0xB_FFFF_FFFF	15GB	Any	PCI-Express memory-mapped IO. The PCI address is {28'b0, 1'b0, cpu_addr[33:0]}. Note 32 bit PCI devices are visible in only the first 4GB of this region; only 64 bit devices are visible in the final 12GB.
0xC_0000_0000	0xC_EFFF_FFFF	~4GB	Any	PCI-Express port-mapped IO. PCI port I/O address = cpu_addr[31:0].
0xC_F000_0000	0xC_FFFF_FFFF	256MB	32-bit	PCI-Express configuration space IO. PCI config address = {cpu_addr[27:16], 4'b0, [11:0]}.
0xD_0000_0000	0xD_FFFF_FFFF	4GB	None	Reserved.
0xE_0000_0000	0xE_7FFF_FFFF	2GB	32-bit	Internal SCB bus registers. This space is further divided into 128 subsections based on the encoding described in AddrSubId. See 16.6.6.
0xE_8000_0000	0xE_FFFF_FFFF	2GB	64-bit	Internal Non-SCB registers. This space is further divided into 128 subsections based on the encoding described in AddrSubId.
0xF_0000_0000	0xF_FFFF_FFFF	4GB	None	Reserved

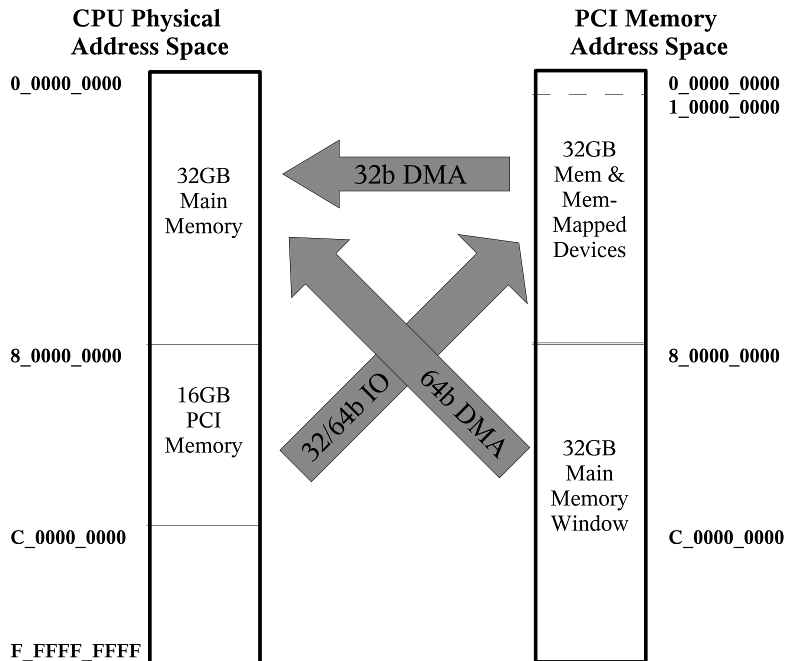


Figure 16.1: Physical CPU to/from PCI addresses

16.4 PCI Address Regions

PCI has three distinct address spaces. PCI Config space and PCI port-mapped IO space are special spaces used for CPU generated transactions, and have no special address decodings. The 64-bit PCI Memory Space is divided into the following regions:

Start Address	End Address	Size	Access	Description
0x0_0000_0000	0x7_FFFF_FFFF	32GB	Any	Maps back to cachable memory, or PCI memory I/O registers, based on a subtractive decode in the PMI. Note only the low 4GB is visible to 32-bit PCI devices, and thus this space may have "holes" to insert the 32-bit devices.
0x8_0000_0000	0xF_FFFF_FFFF	32GB	Any	Maps back to cachable memory. The PMI zeros PCI address bit 35 to generate the memory address. As this region maps all memory without I/O device holes, it should be the DMA region used for all 64 bit PCI devices.
0x10_0000_0000	Rest			Reserved

16.4.1 Software allocation of PCI address space

When allocating addresses for memory-mapped devices on the PCI bus, software needs to exercise caution in the allocation of the addresses. While prefetchable memory must support 64 bit addressing, non-prefetchable may only support 32 bit addressing, which limits devices to the low 4GB of the address space. This suggests the following policy:

1. Allocate BARs starting with the largest request and working down. This avoids holes, as the PCI spec suggests.
2. Allocate 64-bit capable memory BARs anywhere between PCI addresses 4GB and 16GB (Physical addresses 9_0000_0000 and D_FFFF_FFFF).

3. Allocate 32-bit only capable memory BARs working down from FFFF_FFFF to 0. Working from the top-down increases the likelyhood that 32 bit DMA devices will be able to see all of memory.

4. 32-bit DMA devices, if there are any, may see main memory in a window between PCI addresses 0 and the beginning of the first 32-bit BAR allocated in step 3. The rest of memory is inaccessable, and memory copies will be required for DMA to memory outside this window. (High performance devices should be 64-bit, so this shouldn't matter for performance.)

5. 64-bit DMA devices access main memory with PCI addresses 8_0000_0000 to F_FFFF_FFFF, which map down to physical addresses 0 to 7_FFFF_FFFF. All of memory is visible in this window.

16.5 General Behavior

16.5.1 Access size

Software must use the appropriately sized transaction to access registers, using the wrong size results in unpredictable behavior. See 16.3 on page 843 for which areas are 32-bit or 64-bit only.

16.5.2 Read side effects

Unless explicitly specified in a register definition with a "S" in the type field, reads do not have side effects.

16.5.3 Illegal Addresses

Access to addresses that are not implemented (either unspecified or mapping to non-existant memory) will cause unspecified behavior. On writes, this may include a No-Op, aliasing to other addresses, or creation of machine checks. On reads, this may include returning random data, aliasing to a register with read side effects, or creation of machine checks. However, all illegal address accesses will complete, they shall not hang.

16.6 Registers and Definitions

16.6.1 Package Attributes

Package

chip_addr_spec

16.6.2 Definitions

Defines

ADDR

Constant	Mnemonic	Definition
32'd40	PABITS	Physical Address Bits. Number of physical address bits implemente
32'd39	IOBIT	Memory/IO Bit. Address bit that selects memory versus no cachable IO space.
36'h0_1fc0_0000	BOOT	Processor Boot Address. First processor fetch is from this address.
36'h7_2000_0000	BOOT1_PA	Scratch space for boot1 phase.
64'ha000_0007_2000_0000	BOOT1_VA	Scratch space for boot1 phase.
36'h7_2001_0000	BOOT2_PA	Scratch space for boot2 phase.
64'ha000_0007_2001_0000	BOOT2_VA	Scratch space for boot2 phase.
64'hffff_ffff_ff20_0000	EJTAG_FASTDATA_VA	EJtag Fastdata register.
64'hffff_ffff_ff20_0200	EJTAG_BOOT_VA	EJtag Boot address.

16.6.3 Manufacturer Enumeration

AddrTapMfgr specifies the JTAG manufacturer number in the R_SysTapIDecode and R_CpuTapIDCODE registers.

Enum

AddrTapMfgr

Constant	Mnemonic	Definition
11'h2c2	SICORTEX	EJTAG Manufacturer ID for SiCortex. (ID 66, bank 6.)

16.6.4 Product Enumeration

AddrProduct specifies the product name for the R_ScbChipRev (see 10.14.6) and R_CpuPRID registers. It is also used for the JTAG part number in R_SysTapIDecode and R_CpuTapIDCODE register.

Enum

AddrProduct

Attributes

-kernel

Constant	Mnemonic	Product	Definition
8'd19	ICE9		Ice9a for SCX-1000 series. Used in R_CpuPRId, R_ScbChipRev and R_SysTapIDecode registers.
8'd20	ICE9_CPU0		Ice9 EJTAG for CPU0. Used in R_CpuTapIDECODE EJTAG UDR only. This differs from ICE9 above so that we may differentiate each EJTAG TAP from the SysChain TAP.
8'd21	ICE9_CPU1		Ice9 EJTAG for CPU1.
8'd22	ICE9_CPU2		Ice9 EJTAG for CPU2.
8'd23	ICE9_CPU3		Ice9 EJTAG for CPU3.
8'd24	ICE9_CPU4		Ice9 EJTAG for CPU4.
8'd25	ICE9_CPU5		Ice9 EJTAG for CPU5.
8'd26	ICE9B		Ice9b for SCX-1000 series. Used in R_CpuPRId, R_ScbChipRev and R_SysTapIDecode registers.
8'd27	ICE9B_CPU		Ice9b EJTAG part number for CPUs. Used in R_CpuTapIDECODE EJTAG UDR only. This differs from ICE9 above so that we may differentiate each EJTAG TAP from the SysChain TAP. In ICE9B, each processor's UDR is differentiated with the revision number, rather then a different AddrProduct encoding.
8'd30	TWC9A	twc9a+	Twice9A. Used in R_CpuPRId, R_ScbChipRev and R_SysTapIDecode registers.
8'd31	TWC9A_CPU	twc9a+	Twice9A EJTAG part number for CPUs.

16.6.5 Address Bus Stop Numbers

This enumeration contains the software bus stop number, used by the address assignments below, and interrupts. Physical stop numbers may differ without affecting software, see 7.17.10.

Enum

AddrBusStop

Attributes

-kernel

Constant	Mnemonic	Product	Definition
4'h0	COHO		Coherence controller on odd side
4'h1	DMA		DMA controller
4'h2	PS0		L2 segment for processor 0
4'h3	PS1		L2 segment for processor 1
4'h4	PS2		L2 segment for processor 2
4'h5	PS3		L2 segment for processor 3
4'h6	PS4		L2 segment for processor 4
4'h7	PS5		L2 segment for processor 5
4'h8	PCI		Pci controller
4'h9	COHE		Coherence controller on even side
4'hA	PS6	TWC9A+	L2 segment for processor 6
4'hB	PS7	TWC9A+	L2 segment for processor 7
4'hC	PS8	TWC9A+	L2 segment for processor 8
4'hD	PS9	TWC9A+	L2 segment for processor 9
4'hE			Reserved. (Local loopback and aliasing.)
4'hF			Reserved. (Broadcast to all nodes, legal from COHE or COHO only)

16.6.6 Sub-chip IDs

The IO region is split into 128 pieces, one for each major subcomponent on the ICE9. This same encoding determines the upper address bits (30:24) of the control registers in each subchip, and if using the SCB, the SCB identifier. Furthermore, address bits (27:24) or enum bits (3:0) must match the AddrBusStop of that component. For example a AddrSubId of 7'h03 corresponds to SCB address 0xE03xx_xxxx.

The Clk column below indicates what clock domain that SCB slave operates on, if it has a slave. Scb performance counters only count cross-products correctly when comparing events in the same clock domain.

The Events column indicates the enumberation listing performance counter event definitions. See the appropriate sub-chip spec for details.

Enum

AddrSubId
(This table is grouped by bus stop, thus is is sorted by the lower nibble, then upper nibble.)

Constant	Mnemonic	(Clk)	(Events)	Product	Definition
7'h00	COHO	cclk			Odd Coherence Controller.
7'h10	WTIO	n/a			Magic address range used internally by CSW WTIO transactions. See CAC_IO_WTIOADDR define.
7'h20	SIM	n/a			Magic address range for simulator control only.
7'h01	DMA	cclk	DmaScbEvent		DMA Engine.
7'h41	OCTBPS6	cclk		TWC9A+	OCLA Collector block for PS6
7'h51	OCTBPS7	cclk		TWC9A+	OCLA Collector block for PS7
7'h61	OCTBPS8	cclk		TWC9A+	OCLA Collector block for PS8
7'h71	OCTBPS9	cclk		TWC9A+	OCLA Collector block for PS9
7'h02	CPU0	pclk	CpuScbEvent		Processor 0. Note all CPU encodings must be sequentially encoded.
7'h12	CAC0	n/a			L2 Cache 0. (Model directRead/directWrite access only; use CACLOC for registers.)
7'h22	CPU6	pclk	CpuScbEvent	TWC9A+	Processor 6.
7'h32	CAC6	n/a		TWC9A+	L2 Cache 6.
7'h03	CPU1	pclk	CpuScbEvent		Processor 1.
7'h13	CAC1	n/a			L2 Cache 1.
7'h23	CPU7	pclk	CpuScbEvent	TWC9A+	Processor 7.
7'h33	CAC7	n/a		TWC9A+	L2 Cache 7.

Constant	Mnemonic	(Clk)	(Events)	Product	Definition
7'h04	CPU2	pclk	CpuScbEvent		Processor 2.
7'h14	CAC2	n/a			L2 Cache 2.
7'h24	CPU8	pclk	CpuScbEvent	TWC9A+	Processor 8.
7'h34	CAC8	n/a		TWC9A+	L2 Cache 8.
7'h05	CPU3	pclk	CpuScbEvent		Processor 3.
7'h15	CAC3	n/a			L2 Cache 3.
7'h25	CPU9	pclk	CpuScbEvent	TWC9A+	Processor 9.
7'h35	CAC9	n/a		TWC9A+	L2 Cache 9.
7'h06	CPU4	pclk	CpuScbEvent		Processor 4.
7'h16	CAC4	n/a			L2 Cache 4.
7'h07	CPU5	pclk	CpuScbEvent		Processor 5.
7'h17	CAC5	n/a			L2 Cache 5.
7'h08	SCBM	cclk	ScbScbEvent		Serial Control Bus Master. (SCBM's own internal registers, not registers of other subchips on the SCB bus).
7'h18	PCIE	cclk	PmiScbEvent		PCI-Express PMI internal registers. (Not devices ON the PCI bus.)
7'h28	I2C	n/a	n/a		I2C Bus Controller.
7'h38	UART	n/a	n/a		UART.
7'h48	DDR0	dclk	DdrxEvent		SDRAM 0.
7'h58	DDR1	dclk	DdrxEvent		SDRAM 1.
7'h68	OCLA	cclk	n/a		On-chip logic analyzer, common control block.
7'h09	COHE	cclk			Even Coherence Controller
7'h49	OTRBCPS6	cclk		TWC9A+	OCLA Trigger block for PS6
7'h59	OTRBCPS7	cclk		TWC9A+	OCLA Trigger block for PS7
7'h69	OTRBCPS8	cclk		TWC9A+	OCLA Trigger block for PS8
7'h79	OTRBCPS9	cclk		TWC9A+	OCLA Trigger block for PS9
7'h0A	OCTBCOHE	cclk			OCLA Collector block for COHE
7'h1A	OCTBCOHO	cclk			OCLA Collector block for COHO
7'h2A	OTRBCCOHE	cclk			OCLA Trigger block for COHE
7'h3A	OTRBCCOHO	cclk			OCLA Trigger block for COHO
7'h4A	OCTBFSWI	cclk			OCLA Collector block for FSW Inputs
7'h5A	OCTBFSWO	cclk			OCLA Collector block for FSW Outputs
7'h0B	OCTBPS0	cclk			OCLA Collector block for PS0
7'h1B	OCTBPS1	cclk			OCLA Collector block for PS1
7'h2B	OCTBPS2	cclk			OCLA Collector block for PS2
7'h3B	OCTBPS3	cclk			OCLA Collector block for PS3
7'h4B	OCTBPS4	cclk			OCLA Collector block for PS4
7'h5B	OCTBPS5	cclk			OCLA Collector block for PS5
7'h6B	OCTBDMA	cclk			OCLA Collector block for DMA
7'h7B	OCTBPMI	cclk			OCLA Collector block for PMI/BBS
7'h0C	OTRBCPS0	cclk			OCLA Trigger block for PS0
7'h1C	OTRBCPS1	cclk			OCLA Trigger block for PS1
7'h2C	OTRBCPS2	cclk			OCLA Trigger block for PS2
7'h3C	OTRBCPS3	cclk			OCLA Trigger block for PS3
7'h4C	OTRBCPS4	cclk			OCLA Trigger block for PS4
7'h5C	OTRBCPS5	cclk			OCLA Trigger block for PS5
7'h6C	OTRBCDMA	cclk			OCLA Trigger block for DMA Codeword
7'h7C	OTRBVDMA	cclk			OCLA Trigger block for DMA Vector
7'h0D	FLR0	sclk	FlrScbEvent		Fabric Link 0 Receive. (via SCB)
7'h1D	FLR1	sclk	FlrScbEvent		Fabric Link 1 Receive. (via SCB)
7'h2D	FLR2	sclk	FlrScbEvent		Fabric Link 2 Receive. (via SCB)

Constant	Mnemonic	(Clk)	(Events)	Product	Definition
7'h3D	FLT0	sclk	FltScbEvent		Fabric Link 0 Transmit. (via SCB)
7'h4D	FLT1	sclk	FltScbEvent		Fabric Link 1 Transmit. (via SCB)
7'h5D	FLT2	sclk	FltScbEvent		Fabric Link 2 Transmit. (via SCB)
7'h6D	QSC	sclk	n/a		Fabric Link Quad Controller. (via SCB)
7'h7D	FSW	sclk	FswScbEvent		Fabric Switch (via SCB)
7'h1E	CACLOC	n/a	n/a		L2 Local Cache. Local access to control registers for Processor X by Processor X.
7'h2E	INTR	n/a	n/a		Interrupt cycle. Local access by each processor.
7'h3E	SPCL	n/a	n/a		Special cycle. Local access by each processor.
7'h0F	OTRBCPMI	cclk			OCLA Trigger block for PMI/CSW Bus Stop
7'h1F	OTRBVFSWO	cclk			OCLA Trigger block for FSW Vector Output
7'h2F	OTRBVFSWI	cclk			OCLA Trigger block for FSW Vector Input
7'h3F	OTRBCFSW	cclk			OCLA Trigger block for FSW Codeword
7'h4F	OTRBCPMII	cclk			OCLA Trigger block for PMI/BBS Internals

16.6.7 Main Memory Region

Register
R_Mem[0x1_FFFF_FFFF:0]
Address
0x0_0000_0000-0x7_FFFF_FFFF
Attributes
-noregtest -kernel

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Data	RW	x		Main Memory. Transactions to this region will be cached, and misses will go to the SDRAM.

16.6.8 PCI Memory Region

Register
R_PciMem[0x0_FFFF_FFFF:0]
Address
0x8_0000_0000-0xB_FFFF_FFFF
Attributes
-noregtest -kernel

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Data	RW	x		PCI-Express Memory. Transactions to this region will initiate PCI-Express bus Memory reads or writes. Note 32 bit PCI devices are visible in only the first 4GB of this region; only 64 bit devices are visible in the final 12GB.

16.6.9 PCI IO Region

Register
R_PciIo[0x0_3BFF_FFFF:0]
Address
0xC_0000_0000-0xC_EFFF_FFFF
Attributes
-noregtest -kernel

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Data	RW	x		PCI-Express IO Space. Transactions to this region will initiate PCI-Express bus IO reads or writes.

16.6.10 PCI Config Region

Register
R_PciConfig[0x0_03FF_FFFF:0]
Address
0xC_F000_0000-0xC_FFFF_FFFF
Attributes
-noregtest -kernel

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Data	RW	x		PCI-Express Config Space. Transactions to this region will initiate PCI-Express bus config reads or writes.

16.6.11 Internal SCB Region

Register
R_IoScb[0x0_1FFF_FFFF:0]
Address
0xE_0000_0000-0xE_7FFF_FFFF
Attributes
-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Data	RW	x		Internal SCB Registers. Transactions to this region go over the SCB bus to the appropriate sub-chip registers.

16.6.12 Internal Non-SCB Region

Register
R_Io[0x1FFF_FFFF:0]
Address
0xE_8000_0000-0xE_FFFF_FFFF
Attributes
-noregtest

Bit	Mnemonic	Access	Reset	Type	Definition
31:0	Data	RW	x		Internal Non-SCB Registers. Transactions to this region go over the CSW or other busses to the appropriate sub-chip registers.

Chapter 17

Pinout

[Last Modified $Id: chippins.lyx 18812 2006-04-26 17:37:49Z jackson $]

17.1 Overview

This chapter describes the signals, drivers, and pin assignments of the SC-1000. The pinout includes the following major collections of signals:

- Clocks and reset
- 3 input and 3 output SiCortex fabric links
- 2 DDR2 channels
- 8-lane PCI Express port with auxiliary bus
- Console port, serial management bus, JTAG, chip tester scan chains, etc.
- Power and ground

17.2 Signal List

Group	Signal	#	I/O	Type	Description
Fabric 0		**79**			
	flr0_dt_h[7:0]	8	I	f	Fabric 0 inbound data (port a), high differential
	flr0_dt_l[7:0]	8	I	f	Fabric 0 inbound data (port a), low differential
	flr0_fc_h	1	O	f	Fabric 0 inbound data (port a) flow control, high differential
	flr0_fc_l	1	O	f	Fabric 0 inbound data (port a) flow control, low differential
	flt0_dt_h[7:0]	8	O	f	Fabric 0 outbound data (port x), high differential
	flt0_dt_l[7:0]	8	O	f	Fabric 0 outbound data (port x), low differential
	flt0_fc_h	1	I	f	Fabric 0 outbound data (port x) flow control, high differential
	flt0_fc_l	1	I	f	Fabric 0 outbound data (port x) flow control, low differential
	VDDF	20	A	power	1.2V fabric pad voltage
	VSS	23	A	power	Ground
Fabric 1		**79**			
	fl[rt]1_*	36	*	f	Fabric 1 ports b (in) and y (out), similar to fl[rt]0_*
	VDDF	20	A	power	1.2V fabric pad voltage
	VSS	23	A	power	Ground

Group	Signal	#	I/O	Type	Description
Fabric 2		**79**			
	fl[rt]2_*	36	*	f	Fabric 2 ports c (in) and z (out), similar to fl[rt]0_*
	VDDF	20	A	power	1.2V fabric pad voltage
	VSS	23	A	power	Ground
Fabric Miscellaneous		**25**			
	flrx_nc_h	1	O	f	Unused fabric transmit lane, high differential
	flrx_nc_l	1	O	f	Unused fabric transmit lane, low differential
	fltx_nc_h	1	I	f	Unused fabric receive lane, high differential
	fltx_nc_l	1	I	f	Unused fabric receive lane, low differential
	fl_pll_vdd[6:0]	7	A	analog	Fabric quad macro PLL voltage (filtered 2.5V)
	fl_pll_rtn[6:0]	7	A	analog	Fabric quad macro PLL reference return
	fl_rext[6:0]	7	A	analog	Fabric quad macro termination reference resistor
DDR 0		**251**			
	d0_ck_h[2:0]	3	O	sstl1.8	DDR 0 clock, high differential
	d0_ck_l[2:0]	3	O	sstl1.8	DDR 0 clock, low differential
	d0_dm[8:0]	9	O	sstl1.8	DDR 0 data mask
	d0_dqs_h[8:0]	9	B	sstl1.8	DDR 0 data strobe, high differential
	d0_dqs_l[8:0]	9	B	sstl1.8	DDR 0 data strobe, low differential
test-mode overrides see sec. 17.3	d0_dq[63:0]	64	B	sstl1.8	DDR 0 data
test-mode overrides see sec. 17.3	d0_cb[7:0]	8	B	sstl1.8	DDR 0 ecc (alias d1_dq[71:64])
	d0_we_l	1	O	sstl1.8	DDR 0 write enable
	d0_cas_l	1	O	sstl1.8	DDR 0 column strobe
	d0_ras_l	1	O	sstl1.8	DDR 0 row strobe
	d0_cs_l[3:0]	4	O	sstl1.8	DDR 0 chip select ([3:2] NC on PCB])
	d0_ba[2:0]	3	O	sstl1.8	DDR 0 bank address
test-mode overrides see sec. 17.3	d0_ad[15:0]	16	O	sstl1.8	DDR 0 row and column address
	d0_cke[3:0]	4	O	sstl1.8	DDR 0 clock enable ([3:2] NC on PCB])
	d0_odt[3:0]	4	O	sstl1.8	DDR 0 on-die termination control ([3:2] NC on PCB])
	d0_reset_l	1	O	sstl1.8	DDR 0 reset
	D0_VREF	6	A	analog	DDR 0 reference voltage
	d0_rext	1	A	analog	DDR 0 termination reference resistor
	VDDM	7	A	power	2.5V DDR2 receive pad voltage
	VDDR	43	A	power	1.8V DDR2 transmit pad voltage
	VSS	54	A	power	Ground
DDR 1		**250**			
	d1_*, D1_VREF	59	*	*	DDR 1 control, similar to d0_*
test-mode overrides see sec. 17.3	d1_ad[15:0]	16	O	sstl1.8	DDR 1 row & column address
test-mode overrides see sec. 17.3	d1_dq[63:0]	64	B	sstl1.8	DDR 1 data
test-mode overrides see sec. 17.3	d1_cb[7:0]	8	B	sstl1.8	DDR 1 ecc (alias d1_dq[71:64])
	VDDM	7	A	power	2.5V DDR2 receive pad voltage
	VDDR	43	A	power	1.8V DDR2 transmit pad voltage
	VSS	53	A	power	Ground

Group	Signal	#	I/O	Type	Description
PCI Express		**115**			
	pci_tx_h[7:0]	8	O	pcie	PCI-E transmit data, high differential
	pci_tx_l[7:0]	8	O	pcie	PCI-E transmit data, low differential
	pci_rx_h[7:0]	8	I	pcie	PCI-E receive data, high differential
	pci_rx_l[7:0]	8	I	pcie	PCI-E receive data, low differential
	pci_ref_clk_h	1	O	lvds	PCI-E 100MHz reference clock output, high differentia (also test_clk_e_h)
	pci_ref_clk_l	1	O	lvds	PCI-E 100MHz reference clock output, low differential (also test_clk_e_l)
	pci_ref_clk_vref	1	A	analog	PCI-E reference clock output buffer reference voltage
	pci_rext	1	A	analog	PCI-E external reference resistor
	pci_atnled	1	O	cmos, 4mA	PCI-E module attention LED
	pci_pwrled	1	O	cmos, 4mA	PCI-E module power LED
	pci_pwren_l	1	O	cmos, 4mA	PCI-E module power enable
	pci_pwrgd_l	1	I	cmos, pullup	PCI-E module power good (pullup on PCB)
	pci_pwrflt_l	1	I	cmos, pullup	PCI-E module power fault (pullup on PCB)
	pci_prsnt_l	1	I	cmos, pullup	PCI-E module present (pullup on PCB)
	pci_perst_l	1	O	cmos, 4mA	PCI-E module reset
	VDDM	35	A	power	2.5V PCI Express pad voltage
	VSS	37	A	power	Ground
Miscellaneous		**114**			
	sys_uart_txd	1	T	cmos, 4mA	serial port transmit data (open drain output)
	sys_uart_rxd	1	I	cmos, pullup	serial port receive data
	sys_uart_rts_l	1	T	cmos, 4mA	serial port receiver request-to-send output (open drain output)
	sys_uart_cts_l	1	I	cmos, pullup	serial port transmitter clear-to-send input
test-mode overrides see sec. 17.3	sys_i2c_sda	1	B	cmos, 4mA	serial management bus data (open drain output)
test-mode overrides see sec. 17.3	sys_i2c_scl	1	B	cmos, 4mA	serial management bus clock (open drain output)
	sch_trst_l	1	I	cmos	SiCortex test reset (pullup on PCB)
	sch_tck	1	I	cmos	SiCortex test clock
	sch_tms	1	I	cmos	SiCortex test mode select
	sch_tdi	1	I	cmos	SiCortex test data in
	sch_tdo	1	T	cmos, 4mA	SiCortex test data out
	jtag_trst_l	1	I	cmos	JTAG test reset (pullup on PCB)
	jtag_tck	1	I	cmos	JTAG test clock
	jtag_tms	1	I	cmos	JTAG test mode select
	jtag_tdi	1	I	cmos	JTAG test data in
	jtag_tdo	1	T	cmos, 4mA	JTAG test data out
test-mode overrides see sec. 17.3	test_sdi[99:88]	12	B	cmos, 8mA, pullup	Chip tester scan chain serial data in (NC on PCB). These pins either get no boundary-scan insertion or ge observe-only boundary-scan.
test-mode overrides see sec. 17.3	test_sdo[99:88]	12	B	cmos, 8mA, pullup	Chip tester scan chain serial data out (NC on PCB). These pins either get no boundary-scan insertion or ge observe-only boundary-scan.
	test_mode[3:0]	4	I	cmos, pulldown	Chip tester test mode select (bus together and pull up on PCB)
	test_mode_en	1	I	cmos, pulldown	Chip tester test-mode valid (pull down on PCB)
	test_scan_en	1	I	cmos, pulldown	Chip tester scan enable (pull down on PCB)
	sys_clk_e_h	1	I	lvds	66.7MHz system reference clock, high differential
	sys_clk_e_l	1	I	lvds	66.7MHz system reference clock, low differential

Group	Signal	#	I/O	Type	Description
	sys_clk_o_h	1	I	lvds	66.7MHz system reference clock, high differential
	sys_clk_o_l	1	I	lvds	66.7MHz system reference clock, low differential
	test_d0clk_h	1	I	lvds	DDR 0 test clock, high differential (NC on PCB)
	test_d0clk_l	1	I	lvds	DDR 0 test clock, low differential (NC on PCB)
	test_d1clk_h	1	I	lvds	DDR 1 test clock, high differential (NC on PCB)
	test_d1clk_l	1	I	lvds	DDR 1 test clock, low differential (NC on PCB)
	test_iclk_h	1	I	lvds	PCI-E test clock, high differential (NC on PCB)
	test_iclk_l	1	I	lvds	PCI-E test clock, low differential (NC on PCB)
	test_pclk_h	1	I	lvds	Processor test clock, high differential (NC on PCB)
	test_pclk_l	1	I	lvds	Processor test clock, low differential (NC on PCB)
	test_cclk_h	1	I	lvds	Cache test clock. high differential (NC on PCB)
	test_cclk_l	1	I	lvds	Cache test clock. low differential (NC on PCB)
	test_sclk_h	1	I	lvds	Fabric test clock, high differential (NC on PCB)
	test_sclk_l	1	I	lvds	Fabric test clock, low differential (NC on PCB)
	test_clk_o_h	1	O	lvds	Odd-side test clock output for PLL testing, high differential
	test_clk_o_l	1	O	lvds	Odd-side test clock output for PLL testing, low differential
	test_clk_o_vref	1	A	analog	Odd-side test clock output buffer reference voltage
	sys_d0pll_vdd	1	A	analog	DDR 0 domain PLL voltage (filtered 2.5V)
	sys_d0pll_rtn	1	A	analog	DDR 0 domain PLL reference return
	sys_d1pll_vdd	1	A	analog	DDR 1 domain PLL voltage (filtered 2.5V)
	sys_d1pll_rtn	1	A	analog	DDR 1 domain PLL reference return
	sys_ipll_vdd	1	A	analog	PCI-E domain PLL voltage (filtered 2.5V)
	sys_ipll_rtn	1	A	analog	PCI-E domain PLL reference return
	sys_ppll_vdd	1	A	analog	Processor domain PLL voltage (filtered 2.5V)
	sys_ppll_rtn	1	A	analog	Processor domain PLL reference return
	sys_spll_vdd	1	A	analog	Fabric domain PLL voltage (filtered 2.5V)
	sys_spll_rtn	1	A	analog	Fabric domain PLL reference return
test-mode overrides see sec. 17.3	sys_node[4:0]	5	I	cmos	node number (0-26)
	sys_rst_l	1	I	cmos, pullup	Hard reset (from MSP)
	sys_led_l	1	T	cmos, 12mA	Status LED (open drain)
	sys_atn_l	1	O	cmos, 4mA	Attention request (to MSP)
	spare	1	B	cmos, 4mA, pullup	Spare pin (NC on PCB). Internal connection to R_ScbGpio register.
	sys_ocla_trig	1	O	cmos, 12mA	On-chip Logic Analyzer trigger output
	sys_temp_p	1	A	analog	Temperature-sensing diode P terminal
	sys_temp_n	1	A	analog	Temperature-sensing diode N terminal
	VDDM	9	A	power	2.5V miscellaneous pad voltage
	VSS	18	A	power	Ground
Core Power		**160**			
	VSS	80	A	power	Ground
	VDDC	80	A	power	1.0V core voltage
TOTAL	Total	**1152**			

17.3 List of Normal-Mode Signals and Their Test-Mode Overrides

All the following act according to their normal-mode signal names except when {test_mode_en, test_mode[3:0]} = 5'b1000x or when the SYS TAP instruction register is set to serial-scan mode. In those cases the indicated pins take on the function indicated by the test signal names. The serial-scan-mode mux connection column indicates the order in which the individual ATPG scan chains are connected in series to provide the serial-scan-mode function accessed from the SYS TAP.

Several pins in the DDR PHY also take on a secondary test meaning when {test_mode_en, test_mode[3:0]} = 5'b11000, 5'b11001, or 5'b11010. These test modes are for testing the slave DLLs in the DDR PHY. Each slave DLL provides 2 test clock outputs; there are 2 slave DLLs for each byte lane. I've taken a guess as to which of the dq pins will be used for this purpose, but the final decision will come from eSilicon when the DDR PHY is nearer completion.

The test_sdi[99:88] & test_sdo[99:88] pins take on a third set of personalities in test mode 19 for parametric testing of the DDR PHY drive strength and ODT settings and impedance calibration circuit. These are listed in a separate table below. Because data drive onto these pins must be active during boundary scan (for parametric measurements), any boundary scan that is inserted on the test_sdi[99:88] & test_sdo[99:88] pins must be observe-only (or not have B-scan inserted at all).

Normal Signal (chip level)	Test Signal (chip level)	DDR2 PHY IO instance pin	DDR2 PHY instance core side pin	Test-Mode to activate {test_mode_en, test_mode[3:0]}	2'ndary Test Signal (choice of specific dq's is a guess for now, to be finalized when esilicon is ready)	2'ndary Test Mode to activate
d0_dq[4]	test_sdi[0]	dx_dq[4]	test_sdi[0]	5'd16/17		
d0_dq[5]	test_sdo[0]	dx_dq[5]	test_sdo[0]	5'd16/17		
d0_dq[0]	test_sdi[1]	dx_dq[0]	test_sdi[1]	5'd16/17		
d0_dq[1]	test_sdi[2]	dx_dq[1]	test_sdi[2]	5'd16/17		
d0_dq[6]	test_sdo[1]	dx_dq[6]	test_sdo[1]	5'd16/17	test_dllslav_00_tstclk1	5'd22/23
d0_dq[7]	test_sdo[2]	dx_dq[7]	test_sdo[2]	5'd16/17	test_dllslav_00_tstclk2	5'd22/23
d0_dq[2]	test_sdi[3]	dx_dq[2]	test_sdi[3]	5'd16/17		
d0_dq[3]	test_sdo[3]	dx_dq[3]	test_sdo[3]	5'd16/17		
d0_dq[12]	test_sdi[4]	dx_dq[12]	test_sdi[4]	5'd16/17		
d0_dq[13]	test_sdo[4]	dx_dq[13]	test_sdo[4]	5'd16/17		
d0_dq[8]	test_sdi[5]	dx_dq[8]	test_sdi[5]	5'd16/17		
d0_dq[9]	test_sdi[6]	dx_dq[9]	test_sdi[6]	5'd16/17		
d0_dq[14]	test_sdo[5]	dx_dq[14]	test_sdo[5]	5'd16/17	test_dllslav_01_tstclk1	5'd22/23
d0_dq[11]	test_sdo[6]	dx_dq[11]	test_sdo[6]	5'd16/17	test_dllslav_01_tstclk2	5'd22/23
d0_dq[15]	test_sdi[7]	dx_dq[15]	test_sdi[7]	5'd16/17		
d0_dq[10]	test_sdo[7]	dx_dq[10]	test_sdo[7]	5'd16/17		
d0_dq[21]	test_sdi[8]	dx_dq[21]	test_sdi[8]	5'd16/17		
d0_dq[20]	test_sdo[8]	dx_dq[20]	test_sdo[8]	5'd16/17		
d0_dq[16]	test_sdo[9]	dx_dq[16]	test_sdo[9]	5'd16/17	test_dllslav_02_tstclk2	5'd22/23
d0_dq[17]	test_sdo[10]	dx_dq[17]	test_sdo[10]	5'd16/17	test_dllslav_02_tstclk1	5'd22/23
d0_dq[22]	test_sdi[9]	dx_dq[22]	test_sdi[9]	5'd16/17		
d0_dq[23]	test_sdi[10]	dx_dq[23]	test_sdi[10]	5'd16/17		
d0_dq[18]	test_sdi[11]	dx_dq[18]	test_sdi[11]	5'd16/17		
d0_dq[19]	test_sdi[12]	dx_dq[19]	test_sdi[12]	5'd16/17		
d0_dq[28]	test_sdi[13]	dx_dq[28]	test_sdi[13]	5'd16/17		
d0_dq[29]	test_sdi[14]	dx_dq[29]	test_sdi[14]	5'd16/17		
d0_dq[24]	test_sdi[15]	dx_dq[24]	test_sdi[15]	5'd16/17		
d0_dq[25]	test_sdi[16]	dx_dq[25]	test_sdi[16]	5'd16/17		
d0_dq[30]	test_sdo[11]	dx_dq[30]	test_sdo[11]	5'd16/17	test_dllslav_03_tstclk1	5'd22/23
d0_dq[27]	test_sdo[12]	dx_dq[27]	test_sdo[12]	5'd16/17	test_dllslav_03_tstclk2	5'd22/23
d0_dq[31]	test_sdi[17]	dx_dq[31]	test_sdi[17]	5'd16/17		
d0_dq[26]	test_sdi[18]	dx_dq[26]	test_sdi[18]	5'd16/17		
d0_dq[68]	test_sdi[19]	dx_dq[68]	test_sdi[19]	5'd16/17		
d0_dq[69]	test_sdi[20]	dx_dq[69]	test_sdi[20]	5'd16/17		
d0_dq[64]	test_sdo[13]	dx_dq[64]	test_sdo[13]	5'd16/17	test_dllslav_08_tstclk2	5'd22/23
d0_dq[65]	test_sdo[14]	dx_dq[65]	test_sdo[14]	5'd16/17	test_dllslav_08_tstclk1	5'd22/23
d0_dq[66]	test_sdi[21]	dx_dq[66]	test_sdi[21]	5'd16/17		
d0_dq[70]	test_sdi[22]	dx_dq[70]	test_sdi[22]	5'd16/17		

Normal Signal (chip level)	Test Signal (chip level)	DDR2 PHY IO instance pin	DDR2 PHY instance core side pin	Test-Mode to activate {test_mode_en, test_mode[3:0]}	2'ndary Test Signal (choice of specific dq's is a guess for now, to be finalized when esilicon is ready)	2'ndary Test Mode to activate
d0_dq[71]	test_sdi[23]	dx_dq[71]	test_sdi[23]	5'd16/17		
d0_dq[67]	test_sdi[24]	dx_dq[67]	test_sdi[24]	5'd16/17		
d0_ad[15]	test_sdo[15]	dx_ad[15]	test_sdo[15]	5'd16/17		
d0_ad[14]	test_sdo[16]	dx_ad[14]	test_sdo[16]	5'd16/17		
d0_ad[12]	test_sdo[17]	dx_ad[12]	test_sdo[17]	5'd16/17		
d0_ad[9]	test_sdo[18]	dx_ad[9]	test_sdo[18]	5'd16/17		
d0_ad[11]	test_sdo[19]	dx_ad[11]	test_sdo[19]	5'd16/17		
d0_ad[7]	test_sdo[20]	dx_ad[7]	test_sdo[20]	5'd16/17		
d0_ad[8]	test_sdo[21]	dx_ad[8]	test_sdo[21]	5'd16/17		
d0_ad[6]	test_sdo[22]	dx_ad[6]	test_sdo[22]	5'd16/17		
d0_ad[5]	test_sdo[23]	dx_ad[5]	test_sdo[23]	5'd16/17		
d0_ad[4]	test_sdo[24]	dx_ad[4]	test_sdo[24]	5'd16/17		
d0_ad[3]	test_sdo[25]	dx_ad[3]	test_sdo[25]	5'd16/17		
d0_ad[1]	test_sdo[26]	dx_ad[1]	test_sdo[26]	5'd16/17		
d0_ad[2]	test_sdo[27]	dx_ad[2]	test_sdo[27]	5'd16/17		
d0_ad[0]	test_sdo[28]	dx_ad[0]	test_sdo[28]	5'd16/17		
d0_ad[10]	test_sdo[29]	dx_ad[10]	test_sdo[29]	5'd16/17		
d0_ad[13]	test_sdo[30]	dx_ad[13]	test_sdo[30]	5'd16/17		
d0_dq[36]	test_sdi[25]	dx_dq[36]	test_sdi[25]	5'd16/17		
d0_dq[37]	test_sdi[26]	dx_dq[37]	test_sdi[26]	5'd16/17		
d0_dq[32]	test_sdo[31]	dx_dq[32]	test_sdo[31]	5'd16/17	test_dllslav_04_tstclk2	5'd22/23
d0_dq[33]	test_sdo[32]	dx_dq[33]	test_sdo[32]	5'd16/17	test_dllslav_04_tstclk1	5'd22/23
d0_dq[38]	test_sdi[27]	dx_dq[38]	test_sdi[27]	5'd16/17		
d0_dq[39]	test_sdi[28]	dx_dq[39]	test_sdi[28]	5'd16/17		
d0_dq[34]	test_sdi[29]	dx_dq[34]	test_sdi[29]	5'd16/17		
d0_dq[35]	test_sdi[30]	dx_dq[35]	test_sdi[30]	5'd16/17		
d0_dq[44]	test_sdi[31]	dx_dq[44]	test_sdi[31]	5'd16/17		
d0_dq[45]	test_sdi[32]	dx_dq[45]	test_sdi[32]	5'd16/17		
d0_dq[40]	test_sdi[33]	dx_dq[40]	test_sdi[33]	5'd16/17		
d0_dq[41]	test_sdi[34]	dx_dq[41]	test_sdi[34]	5'd16/17		
d0_dq[42]	test_sdo[33]	dx_dq[42]	test_sdo[33]	5'd16/17	test_dllslav_05_tstclk1	5'd22/23
d0_dq[43]	test_sdo[34]	dx_dq[43]	test_sdo[34]	5'd16/17	test_dllslav_05_tstclk2	5'd22/23
d0_dq[46]	test_sdi[35]	dx_dq[46]	test_sdi[35]	5'd16/17		
d0_dq[47]	test_sdo[35]	dx_dq[47]	test_sdo[35]	5'd16/17		
d0_dq[52]	test_sdi[36]	dx_dq[52]	test_sdi[36]	5'd16/17		
d0_dq[53]	test_sdo[36]	dx_dq[53]	test_sdo[36]	5'd16/17		
d0_dq[48]	test_sdo[37]	dx_dq[48]	test_sdo[37]	5'd16/17	test_dllslav_06_tstclk2	5'd22/23
d0_dq[49]	test_sdo[38]	dx_dq[49]	test_sdo[38]	5'd16/17	test_dllslav_06_tstclk1	5'd22/23
d0_dq[54]	test_sdi[37]	dx_dq[54]	test_sdi[37]	5'd16/17		
d0_dq[55]	test_sdi[38]	dx_dq[55]	test_sdi[38]	5'd16/17		
d0_dq[50]	test_sdi[39]	dx_dq[50]	test_sdi[39]	5'd16/17		
d0_dq[51]	test_sdo[39]	dx_dq[51]	test_sdo[39]	5'd16/17		
d0_dq[56]	test_sdi[40]	dx_dq[56]	test_sdi[40]	5'd16/17		
d0_dq[61]	test_sdo[40]	dx_dq[61]	test_sdo[40]	5'd16/17		
d0_dq[60]	test_sdo[41]	dx_dq[60]	test_sdo[41]	5'd16/17	test_dllslav_07_tstclk2	5'd22/23
d0_dq[57]	test_sdo[42]	dx_dq[57]	test_sdo[42]	5'd16/17	test_dllslav_07_tstclk1	5'd22/23
d0_dq[58]	test_sdi[41]	dx_dq[58]	test_sdi[41]	5'd16/17		
d0_dq[63]	test_sdi[42]	dx_dq[63]	test_sdi[42]	5'd16/17		

Normal Signal (chip level)	Test Signal (chip level)	DDR2 PHY IO instance pin	DDR2 PHY instance core side pin	Test-Mode to activate {test_mode_en, test_mode[3:0]}	2'ndary Test Signal (choice of specific dq's is a guess for now, to be finalized when esilicon is ready)	2'ndary Test Mode to activate
d0_dq[59]	test_sdi[43]	dx_dq[59]	test_sdi[43]	5'd16/17		
d0_dq[62]	test_sdo[43]	dx_dq[62]	test_sdo[43]	5'd16/17		
d1_dq[4]	test_sdi[44]	dx_dq[4]	test_sdi[0]	5'd16/17		
d1_dq[5]	test_sdo[44]	dx_dq[5]	test_sdo[0]	5'd16/17		
d1_dq[0]	test_sdi[45]	dx_dq[0]	test_sdi[1]	5'd16/17		
d1_dq[1]	test_sdi[46]	dx_dq[1]	test_sdi[2]	5'd16/17		
d1_dq[6]	test_sdo[45]	dx_dq[6]	test_sdo[1]	5'd16/17	test_dllslav_10_tstclk1	5'd22/23
d1_dq[7]	test_sdo[46]	dx_dq[7]	test_sdo[2]	5'd16/17	test_dllslav_10_tstclk2	5'd22/23
d1_dq[2]	test_sdi[47]	dx_dq[2]	test_sdi[3]	5'd16/17		
d1_dq[3]	test_sdo[47]	dx_dq[3]	test_sdo[3]	5'd16/17		
d1_dq[12]	test_sdi[48]	dx_dq[12]	test_sdi[4]	5'd16/17		
d1_dq[13]	test_sdo[48]	dx_dq[13]	test_sdo[4]	5'd16/17		
d1_dq[8]	test_sdi[49]	dx_dq[8]	test_sdi[5]	5'd16/17		
d1_dq[9]	test_sdi[50]	dx_dq[9]	test_sdi[6]	5'd16/17		
d1_dq[14]	test_sdo[49]	dx_dq[14]	test_sdo[5]	5'd16/17	test_dllslav_11_tstclk1	5'd22/23
d1_dq[11]	test_sdo[50]	dx_dq[11]	test_sdo[6]	5'd16/17	test_dllslav_11_tstclk2	5'd22/23
d1_dq[15]	test_sdi[51]	dx_dq[15]	test_sdi[7]	5'd16/17		
d1_dq[10]	test_sdo[51]	dx_dq[10]	test_sdo[7]	5'd16/17		
d1_dq[21]	test_sdi[52]	dx_dq[21]	test_sdi[8]	5'd16/17		
d1_dq[20]	test_sdo[52]	dx_dq[20]	test_sdo[8]	5'd16/17		
d1_dq[16]	test_sdo[53]	dx_dq[16]	test_sdo[9]	5'd16/17	test_dllslav_12_tstclk2	5'd22/23
d1_dq[17]	test_sdo[54]	dx_dq[17]	test_sdo[10]	5'd16/17	test_dllslav_12_tstclk1	5'd22/23
d1_dq[22]	test_sdi[53]	dx_dq[22]	test_sdi[9]	5'd16/17		
d1_dq[23]	test_sdi[54]	dx_dq[23]	test_sdi[10]	5'd16/17		
d1_dq[18]	test_sdi[55]	dx_dq[18]	test_sdi[11]	5'd16/17		
d1_dq[19]	test_sdi[56]	dx_dq[19]	test_sdi[12]	5'd16/17		
d1_dq[28]	test_sdi[57]	dx_dq[28]	test_sdi[13]	5'd16/17		
d1_dq[29]	test_sdi[58]	dx_dq[29]	test_sdi[14]	5'd16/17		
d1_dq[24]	test_sdi[59]	dx_dq[24]	test_sdi[15]	5'd16/17		
d1_dq[25]	test_sdi[60]	dx_dq[25]	test_sdi[16]	5'd16/17		
d1_dq[30]	test_sdo[55]	dx_dq[30]	test_sdo[11]	5'd16/17	test_dllslav_13_tstclk1	5'd22/23
d1_dq[27]	test_sdo[56]	dx_dq[27]	test_sdo[12]	5'd16/17	test_dllslav_13_tstclk2	5'd22/23
d1_dq[31]	test_sdi[61]	dx_dq[31]	test_sdi[17]	5'd16/17		
d1_dq[26]	test_sdi[62]	dx_dq[26]	test_sdi[18]	5'd16/17		
d1_dq[68]	test_sdi[63]	dx_dq[68]	test_sdi[19]	5'd16/17		
d1_dq[69]	test_sdi[64]	dx_dq[69]	test_sdi[20]	5'd16/17		
d1_dq[64]	test_sdo[57]	dx_dq[64]	test_sdo[13]	5'd16/17	test_dllslav_18_tstclk2	5'd22/23
d1_dq[65]	test_sdo[58]	dx_dq[65]	test_sdo[14]	5'd16/17	test_dllslav_18_tstclk1	5'd22/23
d1_dq[66]	test_sdi[65]	dx_dq[66]	test_sdi[21]	5'd16/17		
d1_dq[70]	test_sdi[66]	dx_dq[70]	test_sdi[22]	5'd16/17		
d1_dq[71]	test_sdi[67]	dx_dq[71]	test_sdi[23]	5'd16/17		
d1_dq[67]	test_sdi[68]	dx_dq[67]	test_sdi[24]	5'd16/17		
d1_ad[15]	test_sdo[59]	dx_ad[15]	test_sdo[15]	5'd16/17		
d1_ad[14]	test_sdo[60]	dx_ad[14]	test_sdo[16]	5'd16/17		
d1_ad[12]	test_sdo[61]	dx_ad[12]	test_sdo[17]	5'd16/17		
d1_ad[9]	test_sdo[62]	dx_ad[9]	test_sdo[18]	5'd16/17		
d1_ad[11]	test_sdo[63]	dx_ad[11]	test_sdo[19]	5'd16/17		
d1_ad[7]	test_sdo[64]	dx_ad[7]	test_sdo[20]	5'd16/17		

Normal Signal (chip level)	Test Signal (chip level)	DDR2 PHY IO instance pin	DDR2 PHY instance core side pin	Test-Mode to activate {test_mode_en, test_mode[3:0]}	2'ndary Test Signal (choice of specific dq's is a guess for now, to be finalized when esilicon is ready)	2'ndary Test Mode to activate
d1_ad[8]	test_sdo[65]	dx_ad[8]	test_sdo[21]	5'd16/17		
d1_ad[6]	test_sdo[66]	dx_ad[6]	test_sdo[22]	5'd16/17		
d1_ad[5]	test_sdo[67]	dx_ad[5]	test_sdo[23]	5'd16/17		
d1_ad[4]	test_sdo[68]	dx_ad[4]	test_sdo[24]	5'd16/17		
d1_ad[3]	test_sdo[69]	dx_ad[3]	test_sdo[25]	5'd16/17		
d1_ad[1]	test_sdo[70]	dx_ad[1]	test_sdo[26]	5'd16/17		
d1_ad[2]	test_sdo[71]	dx_ad[2]	test_sdo[27]	5'd16/17		
d1_ad[0]	test_sdo[72]	dx_ad[0]	test_sdo[28]	5'd16/17		
d1_ad[10]	test_sdo[73]	dx_ad[10]	test_sdo[29]	5'd16/17		
d1_ad[13]	test_sdo[74]	dx_ad[13]	test_sdo[30]	5'd16/17		
d1_dq[36]	test_sdi[69]	dx_dq[36]	test_sdi[25]	5'd16/17		
d1_dq[37]	test_sdi[70]	dx_dq[37]	test_sdi[26]	5'd16/17		
d1_dq[32]	test_sdo[75]	dx_dq[32]	test_sdo[31]	5'd16/17	test_dllslav_14_tstclk2	5'd22/23
d1_dq[33]	test_sdo[76]	dx_dq[33]	test_sdo[32]	5'd16/17	test_dllslav_14_tstclk1	5'd22/23
d1_dq[38]	test_sdi[71]	dx_dq[38]	test_sdi[27]	5'd16/17		
d1_dq[39]	test_sdi[72]	dx_dq[39]	test_sdi[28]	5'd16/17		
d1_dq[34]	test_sdi[73]	dx_dq[34]	test_sdi[29]	5'd16/17		
d1_dq[35]	test_sdi[74]	dx_dq[35]	test_sdi[30]	5'd16/17		
d1_dq[44]	test_sdi[75]	dx_dq[44]	test_sdi[31]	5'd16/17		
d1_dq[45]	test_sdi[76]	dx_dq[45]	test_sdi[32]	5'd16/17		
d1_dq[40]	test_sdi[77]	dx_dq[40]	test_sdi[33]	5'd16/17		
d1_dq[41]	test_sdi[78]	dx_dq[41]	test_sdi[34]	5'd16/17		
d1_dq[42]	test_sdo[77]	dx_dq[42]	test_sdo[33]	5'd16/17	test_dllslav_15_tstclk1	5'd22/23
d1_dq[43]	test_sdo[78]	dx_dq[43]	test_sdo[34]	5'd16/17	test_dllslav_15_tstclk2	5'd22/23
d1_dq[46]	test_sdi[79]	dx_dq[46]	test_sdi[35]	5'd16/17		
d1_dq[47]	test_sdo[79]	dx_dq[47]	test_sdo[35]	5'd16/17		
d1_dq[52]	test_sdi[80]	dx_dq[52]	test_sdi[36]	5'd16/17		
d1_dq[53]	test_sdo[80]	dx_dq[53]	test_sdo[36]	5'd16/17		
d1_dq[48]	test_sdo[81]	dx_dq[48]	test_sdo[37]	5'd16/17	test_dllslav_16_tstclk2	5'd22/23
d1_dq[49]	test_sdo[82]	dx_dq[49]	test_sdo[38]	5'd16/17	test_dllslav_16_tstclk1	5'd22/23
d1_dq[54]	test_sdi[81]	dx_dq[54]	test_sdi[37]	5'd16/17		
d1_dq[55]	test_sdi[82]	dx_dq[55]	test_sdi[38]	5'd16/17		
d1_dq[50]	test_sdi[83]	dx_dq[50]	test_sdi[39]	5'd16/17		
d1_dq[51]	test_sdo[83]	dx_dq[51]	test_sdo[39]	5'd16/17		
d1_dq[56]	test_sdi[84]	dx_dq[56]	test_sdi[40]	5'd16/17		
d1_dq[61]	test_sdo[84]	dx_dq[61]	test_sdo[40]	5'd16/17		
d1_dq[60]	test_sdo[85]	dx_dq[60]	test_sdo[41]	5'd16/17	test_dllslav_17_tstclk2	5'd22/23
d1_dq[57]	test_sdo[86]	dx_dq[57]	test_sdo[42]	5'd16/17	test_dllslav_17_tstclk1	5'd22/23
d1_dq[58]	test_sdi[85]	dx_dq[58]	test_sdi[41]	5'd16/17		
d1_dq[63]	test_sdi[86]	dx_dq[63]	test_sdi[42]	5'd16/17		
d1_dq[59]	test_sdi[87]	dx_dq[59]	test_sdi[43]	5'd16/17		
d1_dq[62]	test_sdo[87]	dx_dq[62]	test_sdo[43]	5'd16/17		
test_sdi[88]					test_dll_MasterAdj[7]	5'd22/23
test_sdo[89]					test_dll_MasterAdj[6]	5'd22/23
test_sdi[90]					test_dll_MasterAdj[5]	5'd22/23
test_sdo[91]					test_dll_MasterAdj[4]	5'd22/23
test_sdi[92]					test_dll_MasterAdj[3]	5'd22/23
test_sdo[93]					test_dll_MasterAdj[2]	5'd22/23

Normal Signal (chip level)	Test Signal (chip level)	DDR2 PHY IO instance pin	DDR2 PHY instance core side pin	Test-Mode to activate {test_mode_en, test_mode[3:0]}	2'ndary Test Signal (choice of specific dq's is a guess for now, to be finalized when esilicon is ready)	2'ndary Test Mode to activate
test_sdi[94]					test_dll_MasterAdj[1]	5'd22/23
test_sdo[95]	D1clkLock			5'd18	test_dll_MasterAdj[0]	5'd22/23
test_sdi[96]					test_dll_Slave0Adj[7]	5'd22/23
test_sdo[97]	D0clkLock			5'd18	test_dll_Slave0Adj[6]	5'd22/23
test_sdi[98]	ClkOutCtrl[1]			5'd18	test_dll_Slave0Adj[5]	5'd22/23
test_sdo[99]	PciRefLock			5'd18	test_dll_Slave0Adj[4]	5'd22/23
test_sdi[99]	ClkOutCtrl[2]			5'd18	test_dll_Slave0Adj[3]	5'd22/23
test_sdo[98]	SclkLock			5'd18	test_dll_Slave0Adj[2]	5'd22/23
test_sdi[97]	ClkOutCtrl[0]			5'd18	test_dll_Slave0Adj[1]	5'd22/23
test_sdo[96]	PclkLock			5'd18	test_dll_Slave0Adj[0]	5'd22/23
test_sdi[95]					test_dll_Slave1Adj[7]	5'd22/23
test_sdo[94]	IclkLock			5'd18	test_dll_Slave1Adj[6]	5'd22/23
test_sdi[93]					test_dll_Slave1Adj[5]	5'd22/23
test_sdo[92]					test_dll_Slave1Adj[4]	5'd22/23
test_sdi[91]					test_dll_Slave1Adj[3]	5'd22/23
test_sdo[90]					test_dll_Slave1Adj[2]	5'd22/23
test_sdi[89]					test_dll_Slave1Adj[1]	5'd22/23
test_sdo[88]					test_dll_Slave1Adj[0]	5'd22/23
sys_node[4]					test_dll_tstctrl[5]	5'd22/23
sys_node[3]					test_dll_tstctrl[4]	5'd22/23
sys_node[2]					test_dll_tstctrl[3]	5'd22/23
sys_node[1]					test_dll_tstctrl[2]	5'd22/23
sys_node[0]					test_dll_tstctrl[1]	5'd22/23
sys_i2c_sda					test_dll_tstctrl[0]	5'd22/23
sys_i2c_scl					test_dll_reset	5'd22/23

More test-mode overrides for the standard I/O block on the West (odd) end of the North (pci) side:

Normal Signal	Test Signal	Test-Mode to activate {test_mode_en, test_mode[3:0]}	Test Signal	Test Mode to activate
test_sdi[88]	ddp_driv_imped[2]	5'd19		
test_sdo[89]	d1_imp_n[1]	5'd19		
test_sdi[90]	ddp_driv_imped[1]	5'd19		
test_sdo[91]	d1_imp_n[3]	5'd19		
test_sdi[92]	ddp_driv_imped[0]	5'd19		
test_sdo[93]	d0_imp_p[1]	5'd19		
test_sdi[94]	ddp_term_read	5'd19		
test_sdo[95]	d0_imp_p[3]	5'd19		
test_sdi[96]	ddp_term300	5'd19		
test_sdo[97]	d0_imp_n[1]	5'd19		
test_sdi[98]	ddp_term150	5'd19		
test_sdo[99]	d0_imp_n[3]	5'd19		
test_sdi[99]				
test_sdo[98]	d0_imp_n[2]	5'd19		
test_sdi[97]				
test_sdo[96]	d0_imp_n[0]	5'd19		

Normal Signal	Test Signal	Test-Mode to activate {test_mode_en, test_mode[3:0]}	Test Signal	Test Mode to activate
test_sdi[95]	d1_imp_p[3] (tie OE on)	5'd19		
test_sdo[94]	d0_imp_p[2]	5'd19		
test_sdi[93]	d1_imp_p[2] (tie OE on)	5'd19		
test_sdo[92]	d0_imp_p[0]	5'd19		
test_sdi[91]	d1_imp_p[1] (tie OE on)	5'd19		
test_sdo[90]	d1_imp_n[2]	5'd19		
test_sdi[89]	d1_imp_p[0] (tie OE on)	5'd19		
test_sdo[88]	d1_imp_n[0]	5'd19		

Chapter 18

Programming Considerations

[Last modified $Id: pguide.lyx 42289 2007-07-24 15:55:03Z wsnyder $]

18.1 Overview

The rest of this document is pretty detailed. While you could probably find all you need to know in the spec, we've attempted to get all the peculiarities relating to programming the chip right here. In all cases, the procedures and rules outlined here are meant as programmer's hints.

18.2 Memory Transactions and Ordering

18.2.1 The Sync Instruction

18.2.2 I-Stream vs. D-Stream Accesses

18.2.3 I/O ordering

I/O writes from a single CPU are processed in strict order within the memory system, but once the writes leave the memory system, there is no longer any guarantee of ordering. For example, a write to an SCB register may not complete (take effect) before a write to a subsequent DMA engine register. To enforce ordering in situations like this, do an I/O read to the SCB register before doing the DMA engine register write (sync is not required).

When sending SPCL operations to the DMA engine, you must issue a SYNC instruction between every pair of SPCLs, or some SPCLs may be lost in the L2 cache.

The DMA engine has a bug (#1991) which can cause RDIOs to return corrupted data when followed immediately by a WTIO from the same CPU. I/O accesses from different CPUs are not affected, and SPCLs are not affected. When it happens, the WTIO overwrites the data before it can be sent back to the core, so the RDIO incorrectly returns the data from the WTIO. To avoid this, either issue a SYNC instruction between the RDIO and WTIO, or be sure to use the RDIO result before issuing the WTIO. All DMA addresses are affected (RA_DmaImem, RA_DmaDmem, RA_DmaAppIface0,1, etc.) except for those in the SCB range (RA_SDma*). The bug has only been observed when DMA is in the process of doing lots of block writes and the CSW is heavily loaded.

18.2.4 D-Stream vs. I/O Operations and Interrupt Delivery

18.2.4.1 I/O read / Block Write interaction

I/O reads can have a hardware interaction with DMA (or PCI) block writes which has a substantial performance impact. If CPU X is doing an I/O read to some device that's really far away, a DMA or PCI BWT to a cache line A which is owned as D-stream by that processor will not complete until the I/O read completes, regardless of whether CPU X has any intention to use line A. Since the DMA engine writes out received packets using BWTs, this can have a meaningful performance impact on DMA latency.

Software which wishes to use the DMA engine in a high-performance manner can prevent this unhappy circumstance by mapping its DMA receive buffers to physical pages which are not present in the caches of any processor

which does I/O reads to far away places. Note that I/O reads to cache-local addresses (e.g. interrupt registers) will never have this interaction, nor will I/O writes of any kind.

The hardware reason that this case occurs is that both I/O reads and BWTs that hit in a local L2 cache require exclusive use of that L2 cache's "might receive data soon" resource, and if the I/O read gets it first, the BWT might have to wait a while.

18.2.5 Oddball Address Spaces and Physical Addressing

18.2.6 Error Traps

18.2.7 Interrupts and Interrupt Handling

18.2.8 Address Aliasing

Processor segment local control registers (RA_CacLoc registers) are assigned addresses in the range 0xE9E000000 to 0xE9E001000. Addresses in the range 0xE9E000000 to 0xE9E000FFF may be decoded such that bits 11 and 10 are ignored. This means that addresses *alias* in this region such that 0xE9E000Cxx, 0xE9E0008xx, 0xE9E0004xx, and 0xE9E0000xx all address the same register. Similarly addresses {0xE9E000Dxx, 0xE9E0009xx, 0xE9E0005xx, and 0xE9E0001xx} and {0xE9E000Exx, 0xE9E000Axx, 0xE9E0006xx, and 0xE9E0002xx} and {0xE9E000Fxx, 0xE9E000Bxx, 0xE9E0007xx, and 0xE9E0003xx} form sets of aliased addresses.

18.3 The DRAM Controllers

18.3.1 Initial Calibration and Setup

One of the steps involved in DDR calibration involves forcing a write or read tp address X to go to DDR (and not get caught in a cache). For the L2, this is done by previously reading two other addresses Y and Zwhich are known to collide with X. The subtle part is that a sync is required after the two setup reads, because part of the job of the reads of Y and Z is to flush X from the L1. Since the CPU processes hits under misses, if Y or Z is a miss and X would have been a hit, we need to sync to make sure Y and Z have evicted X from the L2 before moving on to read it.

18.3.2 On-the-fly ReCalibration

18.3.2.1 Software filtering of impedance calibration settings

The drive & ODT calibration settings for the DDR I/O cells come from the PDDR2CAL cell. This uses a precision resistor on the board to calibrate out process, temperature and voltage variation effects for precise output drive strength (output impedance) and on die impedance termination (ODT). Because the calibration may produce spurious results, hardware is provided to allow for software filtering of the calibration settings before they are applied to the I/O cells.

Here are some portntially important things to know in designing the software filtering algorithm
(These are pasted from email; the formatting isn't pretty, but then, this is Lyx.)

```
 1. Are IMP_P[3:0] & IMP_N[3:0] reset by CAL_RESET? If so, what values do they
    take on at reset?

    Answer: yes, they are reset, and the values in the SuperPhy are the same as
            the SS values in the email below.
            IMP_P[3:0]= 4'b1100
            IMP_N[3:0]= 4'b1001
   The reason is that when you power up, the CSN signal going to the DIMM from
   the ASIC should be an immediate '1', so the SSTL18 buffers must have
   sufficient drive strength under all PVT conditions. This also implies that
   the DRIVE[] values from the core to the SuperPhy for CSN (and CLK) must also
   have an appropriate value as well:
     cti_clk_driv_imped[] <-------------
     cti_addr_driv_imped[]
```

```
    cti_ctrl_driv_imped[] <-------------
    cti_dqs0_driv_imped[]
    cti_dqs1_driv_imped[]
    cti_dqs2_driv_imped[]
    cti_dqs3_driv_imped[]
    cti_dqs4_driv_imped[]
    cti_dqs5_driv_imped[]
    cti_dqs6_driv_imped[]
    cti_dqs7_driv_imped[]
    cti_dqs8_driv_imped[]
    cti_dq_bl0_driv_imped[]
    cti_dq_bl1_driv_imped[]
    cti_dq_bl2_driv_imped[]
    cti_dq_bl3_driv_imped[]
    cti_dq_bl4_driv_imped[]
    cti_dq_bl5_driv_imped[]
    cti_dq_bl6_driv_imped[]
    cti_dq_bl7_driv_imped[]
    cti_dq_bl8_driv_imped[]
  Finally, the ODT in the ASIC should be turned off, which it will be due to
  the resetn effect on, for example, ddo_dqs_roe[0]. Note: clk and CSN have
  these signals permanently turned off in the SuperPhy.

2. For software filtering of IMP_P[3:0] & IMP_N[3:0]:
  - what is the counting sequence as settings cause decreasing impedance?

    Answer:

    For N: 9 is slow, 5 is typ, 3 is fast PVT.
    So, if you have a single part sitting on the bench, operating with some
    fixed voltage, temp, and process, all unchanging, then increasing
    IMP_N[3:0] will decrease the output impedance.

    For P: 12 is slow, 7 is typ, 4 is fast PVT.
    So, increasing IMP_N[3:0] will decrease the output impedance.

  - what are the expected nominal (i.e., TT process, 1.0V, 25C) values?
    Answer: N= 5, P= 7.

  - how much should we expect to see the values change with voltage &
    temperature, i.e., sensitivites in LSBs /mV & /degree-C?

    Answer: would have to do another HSpice sim to find this. But, based on
    the PVT factors of [1.321, 1.185, 1.101], then a coarse answer would be:
    -------------------------------------------------------
    Voltage
    -------------------------------------------------------
    N: (9/5 - 1)* (0.185 / 0.72349)= 20.46%
    i.e. 20.46% change for 100mV delta, or 0.2046% change for 1mV delta.
    ==> 0.2046% * 5 = 0.01023 numeric change / mV.
    ==> "1mV delta" will require changing the setting from:
        5 to 5.01023.

    P: (12/7 - 1)* (0.185 / 0.72349)= 18.26%
    i.e. 18.26% change for 100mV delta, or 0.1826% change for 1mV delta.
    ==> 0.1826% * 7 = 0.012782 numeric change / mV.
```

```
==> "1mV delta" will require changing the setting from:
    7 to 7.012782.

-------------------------------------------------------
Temperature
-------------------------------------------------------
N: (9/5 - 1)* (0.101 / 0.72349)= 11.168%
i.e. 11.168% change for 100C delta, or 0.11168% change for 1C delta.
==> 0.11168% * 5 = 0.005584 numeric change / C.
==> "1C delta" will require changing the setting from:
    5 to 5.005584

P: (12/7 - 1)* (0.101 / 0.72349)= 9.9715%
i.e. 9.9715% change for 100C delta, or 0.099715% change for 1C delta.
==> 0.099715% * 7 = 0.00698 numeric change / C.
==> "1C delta" will require changing the setting from:
    7 to 7.00698
```

18.3.3 DDR Impedance Calibration and Bug 2013

See Section 8.4.8.36 for a discussion of the different auto calibration modes. Note that CalMode 2 is not currently supported. If any memory transaction is in flight at the time an autocalibration in mode 2 is initiated, the autocal state machine will hang and prevent completion of the calibration loop and thus completion of the memory reference.

18.4 Initializing the PMI/PCI Controller

18.4.1 Unused PCI Controllers

18.4.2 PCI Controllers With Connected Devices

18.4.3 PCI Controllers With No Connected Device

Chapter 19

Differences, Bugs, and Enhancements

19.1 Overview

This chapter summarizes the product differences and errata for the different SiCortex chips. See the corresponding chapters for more information.

19.2 User Code

19.2.1 Product and Chip Pass Differences

1. ICE9B fixes bug2619 whereby **ICE9A requires double load-linkeds** to insure atomicity. This also removes the rationale for the suggestion in bug2807 that R_CpuConfig_LLTIME be programmed to 1 or greater to allow enough time for most atomic sequences to complete; _LLTIME may now be programmed to zero.
2. ICE9B1 fixes bug2826 whereby Multiply Double and friends may get a incorrect results when not followed by a idle cycle, or after write-after-write stalls. This afflicted madd.d, msub.d, mul.d, nmadd.d, nmsub.d, recip.d, rsqrt.d, and sqrt.d.
3. NEED IMPL: TWC9A adds more CPU cores, for a total of 10.
4. **TWC9A uses a new core, IceT. This is described in a different document.**

19.2.2 Known Bugs and Possible Enhancements

1. None.

19.3 Processor Core

19.3.1 Product and Chip Pass Differences

1. ICE9B returns a different product (ICE9B) when reading **R_CpuPRId** and R_CpuTapIDCODE.
2. ICE9B fixes bug1965 whereby **R_CpuErrCtl** reads swap bits 31 and 28. In ICE9A any read-modify-writes need to swap these bits before writing them back.
3. ICE9B improves **micro DTLB performance** bug 2200 with a entry size of 64KB when the corresponding TLB entry is 64KB or larger. If the TLB entry is 16KB, the old 4KB uTLB entry size is used.
4. ICE9B improves probe performance by using 64 byte probes, see bug2202.
5. ICE9B removes an unnecessary syncronizer on the cac_cpu_int wires, this reduces interrupt latency by one pclk.
6. ICE9B adds **performance counter events** for L2 misses and floating point operations, and allows all events to be visible to both counter 0 and counter 1.

7. TWC9A returns a different product (TWC9A) when reading **R_CpuPRId** and R_CpuTapIDCODE.

8. **TWC9A uses a new core, IceT. This is described in a different document.**

19.3.2 Known Bugs and Possible Enhancements (M5KF only)

1. On D-Cache ECC errors, **R_CpuCacheErr_EW** may record the incorrect way number and index**,** see bug1575. As a workaround, software should flush the entire cache on ECC errors.

2. On filling the TLB with a **4KB page**, we should pull a machine check, as 4KB pages are not supported.

3. On **writes to accelerated space**, we should pull a machine check, as they are not supported.

4. We should add a 64-bit cycle counter which is NOT writable, as the current count register is occasionally overwritten by the kernel, bug3342.

5. We should implement the RDHWR instruction so user space code can see the cycle counter and processor number.

6. We should add more VA bits, to enable the VA to be unique across the entire system.

19.4 Addressing

19.4.1 Product and Chip Pass Differences

1. TWC9A adds some values to the AddrBusStop enumeration to support the additional cores, bug3377 .

19.5 L2 Cache

19.5.1 Product and Chip Pass Differences

1. TWC9A's L2 cache is part of the new IceT core, and is described in a different document.

2. TWC9A adds the CswStopNumTwc and CswTidTwc enumeration to support more cores, and more TIDs per core, bug3377.

3. NEED IMPL: TWC9A fixes the R_CacxIntCr[#]_Overflow bit being mis-cleared when clearing R_CacxIntCr[#]_Active, bug3165.

4. NEED IMPL: The R_CohxEccMode_CorEna bit must be set whenever the ICE9 caches are active, bug1990.

5. NEED IMPL: TWC9A pushes IO writes instead of using a special command, bug4898.

6. NEED IMPL: TWC9A removes SPCL in favor of IO writes, bug4899.

7. NEED IMPL: TWC9A stalls issuing probes to avoid large per-cpu probe queues.

19.5.2 Known Bugs and Possible Enhancements

19.6 Memory Controller

19.6.1 Product and Chip Pass Differences

1. ICE9B fixes the DDR unit to support IO driver calibration before the DRAM initialization sequence, bug2276. In ICE9A the Ddr/Ddp units currently only support updating values into the IMP_P_HV[3:0] and IMP_N_HV[3:0] inputs of the DDR2 IO cells during one of the mission mode time CalModes. When SoftReset is asserted the PHY puts in default strong values (low impedence biased) into these.

2. ICE9B fixes some of the ODT on/off range values, bug2401. The NWL controller was supposed to support the following range of ODT turn on/off times for Ice9a's DDR-Phy: ON time range: controlled by DdrxPhyCfg2_AsicDqsOdtOn and DdrxPhyCfg2_AsicDqOdtOn -2.5 clocks <-> 0 clocks (in half cycle increments) relative to the start of the read preamble OFF time range: controlled by DdrxPhyCfg2_AsicDqsOdtOff and DdrxPhyCfg2_AsicDqOdtOff -1.5 clocks <-> 2 clocks (in half cycle increments) relative to the start of the read preamble. However, the bug causes the -2.5 and -2 clocks turn on times to NOT work with turn off times of 1.5 and 2 clocks.

3. TWC9A fixes access to any SCB bus slave hanging while the DDR controller is in reset, bug2928.

4. NEED IMPL: TWC9A drops support for unbuffered DIMMs.

19.6.2 Known Bugs and Possible Enhancements

1. Calibration Mode 2 can cause Ddi to hang waiting for Powerdown, see bug2013. When setting AutoCalUpdate in cal mode 2 (update during prechargePowerdown), the Ddi can hang. This is caused when a request is at the head of the queue requesting to be sent to the controller at the time we start the calibration update process. The calibration logic spins in place waiting for powerdown entry. However, this pending request causes the powerdown counter to be cleared on every cycle, which blocks the Ddr from ever entering powerdown mode. To workaround, do not use calibration mode 2.

2. The DDR bank address could be changed to better optimize page hits, bug2068.

19.7 PCI

19.7.1 Product and Chip Pass Differences

1. ICE9B fixes legacy interrupt D behavior incorrect during a link down, bug1984. In ICE9A if an ASSERT_INTD message arrives from the endpoint, software will service the interrupt. During this time, if the link goes down, an implicit DEASSERT_INTD should occur, but this did not happen. So if the interrupt service routine ends with a "wait for DEASSERT_INTD", and it is possible that it will hang forever.

2. ICE9B fixes ecc error ignored when CLEAR comes at the same time, bug2028. In ICE9A if an ECC error is in effect and the interrupt is raised. Some time software clears the interrupt and an ECC error comes at the same time (in PMI where is checks, or not checks, for ecc error and clear), PMI ignores the second ECC error.

3. ICE9B fixes the MsiBaseAddr register addressing, bug2097. In ICE9A, software has to program the PMI MsiBaseAddr register with an Ice9 address converted into a PCIe space address (look at the address mapping in the hardware spec).

4. ICE9B fixes RX detection not being completed when some lanes are disabled, bug2113. In ICE9A, when one or more lanes of a multi-lane link are disabled using TxCompliance/TxElecIdle as described in Section 8 of the PIPE specification, initiating a receiver detection sequence will cause the PCS layer to hang due to the "turned off" lanes not performing the receiver detection operation. To workaround, enable all lanes prior to performing a receiver detection operation, as lanes which are turned off will not participate in the receiver detection sequence.

5. NEED IMPL: TWC9A fixes only the bottom 16 bit being writable in R_PmiVmReqDat, bug2760. We couldn't find any PCIe vendor which uses vendor messages, so this is of only minor concern.

19.7.2 Known Bugs and Possible Enhancements

1. None.

19.8 DMA

19.8.1 Product and Chip Pass Differences

1. NEED IMPL: TWC9A records the address and syndrome of DRAM ECC errors, bug2157.

2. NEED IMPL: TWC9A fixes generation of bad ECC when ECC correction disabled and a 32-bit aligned packet is read, bug2396. R_SdmaEccMode bit 6 (CifCorrEna) enables ECC correction in CIF. This logic is only needed when the microengine does a BRD from a memory address with bit 2 set (32-bit realignment). When CifCorrEna is off and the microengine does a BRD from a memory address with bit 2 set, the ECC written into the DMA's internal memory (TX or COPY port packet buffer) is incorrectly forced to zero. Data with corrupted ECC may reach the FSW or main memory when the packet is sent. To workaround, leave CifCorrEna always set.

3. NEED IMPL: TWC9A fixes non-correction of ECC during 32-bit realignment operations, bug2403. When the CifCorrEna bit is on, and DMA is doing a read with 32-bit realignment, and there is a single bit error on the data from the CSW, the RTL does not correct the error. The RTL corrects the error inside the DmaCifDatacalg modules, but then incorrectly puts out the uncorrected data on cif_xxx_Data*[63:0] and into the next DmaCifDatacalg module. But the ECC bits on cif_xxx_data*[71:64] are the ECC consistent with the corrected data, so the resulting data appears to have just a single bit error. Workaround: None needed, as the error will be corrected at the destination of the DMA engine.

4. NEED IMPL: TWC9A might double the size of the instruction memory, bug3390.

5. NEED IMPL: TWC9A removes SPCL in favor of IO writes, bug4899.

6. NEED IMPL: TWC9A removes 32 byte writes to support DDR x4 parts, bug4793.

7. MIGHTFIX: TWC9A might fix a performance issue which requires a dead cycle between DMA packets headed into the FSW, bug597.

8. MIGHTFIX: TWC9A might fix DmaCif RDIO being corrupted by subsequent WTIO from the same core, bug1991. This can cause RDIOs to return corrupted data when followed immediately by a WTIO from the same CPU. I/O accesses from different CPUs are not affected, and SPCLs are not affected. When it happens, the WTIO overwrites the data before it can be sent back to the core, so the RDIO incorrectly returns the data from the WTIO. To avoid this, either issue a SYNC instruction between the RDIO and WTIO, or be sure to use the RDIO result before issuing the WTIO. All DMA addresses are affected (RA_DmaImem, RA_DmaDmem, RA_DmaAppIface0,1, etc.) except for those in the SCB range (RA_SDma*). The bug has only been observed when DMA is in the process of doing lots of block writes and the CSW is heavily loaded.

9. MIGHTFIX: Various possible microinstruction enhancements, bug3392, bug3393, bug3394, bug3395, bug3396.

19.8.2 Known Bugs and Possible Enhancements

19.9 Fabric Links

19.9.1 Product and Chip Pass Differences

1. NEED IMPL: TWC9A fixes certain noise patterns from causing fabric deadlocks, bug2132.

2. NEED IMPL: All FL internal counters' increment signals should be wired into the SCB counters, bug3488.

19.9.2 Known Bugs and Possible Enhancements

1. Force retraining should always complete, and software shouldn't have to detect and implement retries.

2. The out-of-band path was never used by software, and could be removed for simplicity if desired.

19.10 Fabric Switch

19.10.1 Product and Chip Pass Differences

1. None.

19.10.2 Known Bugs and Possible Enhancements

1. The FSW has an architectural performance limit preventing 4 ford packets at max rate, bug1832.

19.11 SCB

19.11.1 Product and Chip Pass Differences

1. ICE9B returns a different product (ICE9B) and/or revision (ICE9A1 vs ICE9A0) when reading R_ScbChipRev.
2. ICE9B has reduced latency accessing the SCB's own registers.
3. ICE9B adds a interrupt/attention for when the Chip<->Msp channel is ready for transmit.
4. ICE9B adds R_ScbDInt to replace the SysChain R_SysTapDint register, see bug2223.
5. TWC9A returns a different product (TWC9A) and/or revision when reading R_ScbChipRev.
6. NEED IMPL: TWC9A supports 64 bit SCB slaves and 64 bit registers, see bug4619.
7. TWC9A adds R_ScbDInt_SendDInt6, R_ScbDInt_Cpu6DM, R_ScbAtnInt_Cpu6DMMask, and R_ScbAtnInt_Cpu6DM to support CPUs 6-9.
8. TWC9A fixes reads to fast DDR clock registers returning the wrong results after a CCLK register read, bug4331. Earlier chips required a dummy read between such read sequences.
9. TWC9A will skip sampling bucket pairs where R_ScbPerfBuckets_Event == AllEvent_INVALID. This is backward compatible with other products, which should use that encoding for invalid buckets. bug4265.

19.11.2 Known Bugs and Possible Enhancements

1. In ICE9A and ICE9B, all SCB accesses must be done with 32-bit accesses. Using a 64-bit read/write to access them will put return/write data in the wrong half of the quadword, not simply return or write half of the data.
2. Decouple the SCB CPU#_P[01] events from the CPU performance counter domain (U/S/K), perhaps with new domain bits.
3. SCB performance counts from Ocla TrbC blocks depend on the TrbC configuration, this could be simplified. bug1717.
4. R_ScbPerfEna should have a way to stop immediately, without corrupting, for interrupt handlers. Perhaps add a _Pause bit that stops on current bucket and partial interval. We'll also need to make the partial interval programmable so context switches can reprogram it.
5. R_ScbPerf* registers should be writable without needing to stop sampling.
6. R_ScbInt should indicate what bucket(s) have caused the overflow, to save software from having to read the entire count ram on each overflow, bug2164.
7. R_SysTapMsp transactions should be double buffered, as the Msp decision loop is quite slow.
8. R_ScbInt like most of the other blocks in the chip contains the interrupt state before masking. This requires the interrupt handler to read (or cache) R_ScbIntMask before dispatching interrupts.

19.12 LBS

19.12.1 Product and Chip Pass Differences

1. ICE9A1 returns a different revision (ICE9A1 vs ICE9A0) when reading the IDECODE register.
2. ICE9B fixes Sms Reset syncronized to the wrong clock, bug2055. This required the smsclock to be turned off whenever we wiggle reset, then turned on again a bit later.
3. ICE9B eliminates R_SysTapDint, replaced with the SCB-space R_ScbDInt, bug2223.
4. ICE9B supports transmit interrupts for R_SysTapAtnMsp, and separates RW1C bits, bug2222.
5. NEED IMPL: TWC9A changes the default value for R_SysTapPll_D*clkDifv to support a processor default clock frequency of *FIX* MHz, bug3384.
6. TWC9A fixes access to any SCB bus slave hanging while the DDR controller is in reset, bug2928.
7. TWC9A adds an R_SysTapReset_Lac and _Pmi to separate the R_SysTapReset_Scb bit from also controling the BBS/PMI reset, bug2929. Earlier products needed caution when maintaining FSW/FL traffic during partial reboots.
8. NEED IMPL: TWC9A adds R_SysTapReset_Proc6, and _ProcSms6 to support the additional cores.
9. TWC9A uses R_SysTapInstrTwc instead of R_SysTapInstReg to support the additional cores.
10. TWC9A adds R_SysTapScb64 to access doubleword SCB registers. Code should use this new registers or 64 bit SCB registers will not be visible.
11. NEED IMPL: TWC9A adds R_SysMemInit register and associated functions for on-chip memory initalization. In previous products BIST was used to initalize on-chip memories.
12. NEED IMPL+SPEC: TWC9A will merge the SysChain and E-Silicon chain on-chip instead of off-chip.
13. NEED IMPL+SPEC: TWC9A will replace or make the E-Silicon chain IEEE compliant (on the correct edges).

19.12.2 Known Bugs and Possible Enhancements

1. [Larry] Add a new LBS+SCB region. The msp could set the start address in 32 or 64 bit steps, and then scan in, say 128 bytes with a continuous shift on the scan. Then, while the ice9 digests that block, the msp scans in 128 bytes into the alternate half of the block. This is essentially a block of shared memory accessed on the ice9 side by scb and on the msp side by efficient scan. The scan chain would shift in a direction compatible with the qspi as well. This shared area would be used instead of fastdata (since it would be much faster) for boot2 loading, and we would also use it for block transfers of attn data instead of doing that 26 bits at a time via the current attn register.

19.13 UART

19.13.1 Product and Chip Pass Differences

1. FIX NEED IMPL: TWC9A removes the UART flow control signals. They were never used on the ICE9 modules.

19.14 OCLA

19.14.1 Product and Chip Pass Differences

1. ICE9B fixes GO->0 should shut OFF collection, bug2246. CollectTrace can be left ON by stopping an OCLA program that had not yet seen it's trigger. CollectTrace can only be controlled by a running OCLA program, so you can't shut it off by SCB writes. While CollectTrace is ON, you cannot dump any CTBs. Workarounds: (a) A Diagnostics Dash script has been written that loads and runs a minimal OCLA program to shut off

CollectTrace. (b) The OCLA dump program has been written to detect CollectTrace=ON, and exit with meaningful error message. (c) OCLA Dash scripts and all example OCLA programs have been written with a âgraceful exitâ option, where a specific register-write tells it to shut CollectTrace OFF and stop watching for the trigger it didn't get yet.

2. ICE9B adds new INCRBTH Opcode, bug2179. In ICE9A, although OCLA has 2 counters, you cannot count 2 events concurrently, because if both happen on same clock there's no way to increment both counters.

3. ICE9B enlarges counters from 12 to 16 bits, bug2244.

4. ICE9B fixes PMI CTB ExtMuxSel wired to TRBC, bug1959. The ExtMuxSel wires of OCLA PMI CTB were wired to the SCB register that's supposed to control OCLA PMII TRBC. To workaround, write desired PMI CTB ExtMuxSel value to ExtMuxSel field in control register for PMII TRBC. Fortunately, PMII TRBC has no ExtMux, so this field is otherwise unused. Simplest solution without determining whether you have Ice9A or ICE9B is write desired PMI CTB ExtMuxSel value to both ExtMuxSel fields.

5. ICE9B fixes CAC trigger PrbState obscured by WtPrb2L2, bug1995. OCLA CAC TRBC mux=2 signals PrbState[2:0] had WtPrb2L2 OR-ed into PrbState[2]. To workaround, don't use PrbState as a trigger, or only trigger on PrbState groups of state that you can identify with bits [1:0].

6. ICE9B fixes CAC trigger W0Hit/W1Hit instead of W0Miss/W1Miss, bug2243. In ICE9A, both CAC Trigger Block and Collector Block hookups: (a) Change W0Miss/W1Miss to something better, perhaps W0Hit/W1Hit. Miss is including Idle and I/O. (b) Adjust flops so W0Hit/W1Hit in same clock with related signals. To workaround, (a) qualify with not-Idle and not-IO. (b) Separately feed Hit and the other signals to LAC in separate triggers, then align them with Dly regs in LAC.

7. MIGHTFIX: TWC9A might fix OCLA to SCB uses LAC triggers, bug1717. Passing OCLA events from trigger blocks to SCB Counters ties up LAC trigger configuration, usually preventing simultaneous OCLA use for other purposes. To workaround, accept that you are tying up OCLA with this. The cross connections between OCLA and SCB counting may not be used that much. You might prefer to count SCB events in SCB counters, and count OCLA events in OCLA counters.

8. MIGHTFIX: TWC9A might allow trigger delays for blocks located in other than the CCLK domain, bug1854.

9. MIGHTFIX: TWC9A might add capture mux settings for the CPU program counter and L2<->L1 signals.

10. NEED IMPL: TWC9A might add capture mux settings for the FSW links 1 and 2, bug2232.

11. MIGHTFIX: TWC9A might fix DMA CTB qualifier in wrong clock, bug2193. In DMA's hookups to OCLA, the ue_xxx_DbgValid_c2a signal is sent into the trigger block and CTB, when really it should be delayed by two more cycles. In the CTB as a qualifier we pretty much cannot use it, because you want to use it in combination with other signals like DbgThread_c4a and DbgPc_c4a. To workaround, only do un-qualified collection in DMA CTB. In DMA trigger block, send it and other signals separately on the 2 triggers to LAC, where the Dly regs can align them.

12. MIGHTFIX: TWC9A might add a WtAddr sticky overflow bit, bug2207.

13. MIGHTFIX: TWC9A

19.14.2 Known Bugs

1. Overflow bits still set as OCLA starts, bug1825. OCLA's automatic clearing of counter overflow bits when you start LAC program is delayed a clock or two. Early instructions in LAC program can falsely trigger on overflow depending on the previous use of OCLA. To workaround, never branch on Counter Overflows in first 2 instructions of any LAC program.

2. C CTB WtAddrClr triggered by any address in CTB, bug2026. Writing 0x10 to any SCB register address in a particular Ocla CTB can trigger WtAddrClr (clear write address reg). This even includes unused addresses within the SCB address space of a CTB. To workaround, never write any of the read-only registers.

19.14.3 Possible Enhancements

1. Make both LAC counters 32-bit (currently 16-bits plus sticky overflow bit). There's only one instance of the LAC, so this is very affordable. We've wanted bigger counters when writing LAC programs, and unanticipated but valuable use of OCLA as a highly-configurable counter would benefit from full 32-bit counters.

2. Separate "GO" Register. When you write OCLA management software for one of Ice9's embedded processors, or for the external SSP, you tend to write one function that configures OCLA ahead of time, and another function to tell OCLA to "GO" at roughly the right moment. Currently the GO bit shares register R_LacCtl with some configuration fields that need to be written correctly for what you want OCLA to do. This contributes to messy software design in that you must have handy the values to write to those fields when you write a 1 to GO to start the LAC program. It would be nice if all OCLA configuration could be encapsulated in, and completed by an OCLA configuration function.

3. If SCB reg addresses are cheap, consider breaking R_LacCtl into 3 or 4 registers by type of access, making software easier to write.

4. Collect ON/OFF by Register Write. Provide a super-simple alternative to writing a LAC program, for when exact timing of collection is not critical. Provide one or two registers that allow you, by SCB register write alone, to turn on and off CollectTrace to the CTBs. This allows someone with minimal knowledge of OLCA to quickly collect some trace information and read it out, just by doing easy-to-understand SCB writes and reads. Some semi-steady-state activities can be viewed at an arbitrary time, or you could try more than once till you see it. Or, for more accuracy, you could have Ice9 embedded processor code trigger collection at roughly the right time, and rely on the 1024-entry size of the CTBs to give you a pretty big window to land in. These reg writes would the same logic as the SETCOLL and CLRCOLL opcodes from LAC.

5. Trigger by Register Write. There are ways to do this now, but they're a little obscure. I'm suggesting a very-simple up-front way to trigger your LAC program by writing an SCB register in LAC who's sole purpose is to do this. Aggregate Mask and Match bits 0 and 1 are available, so why not have them driven directly from such a register.

6. Clarify When CTB Has New Contents. Currently it's a little hassle to do sanity checks that your CTB really got new contents from running your LAC program. Especially when you are wondering if you configured everything correctly. You can "trust that a good-status completed LAC program means you have new CTB contents". You can alternate the CTB's external mux between what you want to collect and something else, then read-out the CTB and see that contents changed.

7. CTB Zeroing. An SCB-register "ClearCtb" action-bit in each CTB, that would zero-out the CTB (taking 1024 clocks). This bit could be readable and self-clears after the 1024 clocks have passed, so it's safe to start a new collection.

8. StopOnFull Final Address. Currently, in StopOnFull mode, when the CTB gets full and stops collecting, the final address is 0x000, which is the same address it would have if it never started! Either change this to stop at 0x3FF, or have a sticky overflow bit which clears when you write WtAddrClr in R_CtbxColCtl.

9. StoppedOnFull Status Bit. If in StopOnFull mode, have a read-only bit StoppedOnFull in R_CtbxColCtl. This signal already exists in the CTB Verilog code.

10. Fix the "Collecting" Status Bit. Bit "Collecting" of R_CtbxColCtl is directly flopped off of lac_ctb_CollectTrace_c0a, which means it doesn't take into consideration a CTB in StopOnFull mode that has become full. Reading of the CTB works in that case. Change Collecting to be false if StopOnFull and full. A signal with this info is available in the CTB verilog code. You might also consider having "Collecting" read back as 0 when EnableCollect==0. To be able to see the level of signal lac_ctb_CollectTrace_c0a clearly in one central place, add read-only bit "CollectTrace" to R_LacCtl (or if R_LacCtl gets broken-up into several registers as suggested, put this bit in whatever register contains the other read-only fields).

11. Have 0xFFFFFFFF Indicate Bad Read. If you try to read the contents of your CTB when you cannot, you currently get all-zeros. All-zeros can mean you never collected anything, and also for some units it's a likely read-result if you collected during an idle time. A tiny change in the verilog could make it return 0xFFFFFFFF's for reads when you can't read the CTB. This would be clearly different than a failure to trigger collection, and is an almost-impossible long series of values for any CTB to collect.

12. Stopping LAC Stops Collection. Have a transition of the GO bit 1 -> 0 cause the CLRCOLL action. This eliminates the hazard of someone stopping the LAC program manually by clearing the GO bit, but then being unable to read any CTB contents because CollectTrace is still ON. Have this be by 1 -> 0 transition, not by GO==0, so we can have the previously-mentioned registers that turn on and off collection. The way OLCA is now it can be very irritating if you happened to shut off LAC by writing 0 to the GO bit when collection was ON. There's no straightforward way to shut off collection of all enabled CTBs by register-write, you can only shut them off by opcode CLRCOLL in a running LAC program. This is no problem when the next LAC program you wish to run is of the CTB StopOnFull=0 unqualified style, but if you are doing qualified collection with StopOnFull=1 and you want to start at CTB address=0 it can be a problem. You might think you could just begin every LAC program with a CLRCOLL and your problems would be solved, but there's no way inside a LAC program to clear a CTB's WtAddr.

13. Move Delay Registers into the Trigger Blocks. Having the Delay Registers centralized in LAC means they're all flopped in cclk domain. FSW triggers and trigger blocks are in sclk domain. To be able to line-up FSW signals into a complex trigger is hard, although this was partly solved by providing some FSW trigger signals to it's trigger blocks more than once, with different sclk delays. The best solution to this is to have the delay registers in the Trigger Blocks, not centralized in the LAC.

14. More External-Mux Values, or Extra Mux in FSW. Boost the number of bits to control external muxes from 3 to 4 or 5. Do this for all types of trigger and collector blocks. Almost no extra logic is created by this except in those blocks where the extra external-mux-value options are used. The motivation for this is with regard to the Link side of FSW. Currently OCLA in FSW only looks at FLR-0 and FLT-0 signals, due to mux-value limitations. For better board and system debug, to use OCLA freely to see damaged traffic arriving any one particular link, we really want all 6 links covered by OCLA. (b) Another way to get all 6 Link interfaces in FSW into OCLA, without changing OCLA Trigger or Collector blocks, is to just put a new register into FSW. This register in FSW's register address space would take values of 0, 1, or 2, and would drive a first level of muxing, selecting which link-number provides FLR and FLT signals to the current OCLA-register-driven external muxes.

15. More Collection Qualifiers. CTBs currently allow up-to 2 Qualifier signals. In some uses of CTBs there were more signals that would be handy to have available as qualifiers. The external mux selecting data for a CTB often selects between a good number of unrelated interfaces. In a number of cases you just accept that you have to do un-qualified collection, because the 2 qualifiers provided are not relevant to the interface or signals you are looking at.

16. More CTB Qualifier Inputs. Perhaps 4.

17. Use External Mux on Qualifiers. When instantiating CTBs, follow the example of how FSW Vector Trigger Blocks are instantiated, where the external mux selectors vary both the data ***and*** the qualifier to be used.

18. Eliminate Qualifiers in Codeword Trigger Blocks. The way Codeword Trigger Blocks work, all the trigger inputs are effectively qualifiers on each other. There's no reason to handle some inputs differently and call them "qualifiers".

19. Widen Vector Trigger Blocks to 64-Bits. FSW is really the only place where Vector Trigger Blocks are used, because the way they're used in DMA is more naturally served by Codeword Trigger Blocks. In FSW the natural width of the busses looked-at is 64 bits. It would be a usage simplification if the Trigger Block just looked at the 64 bits.

Index

AddrMfgr, 845
AddrProduct, 846

BWTGO, 397

Commands
- BRD, 407–409, 411–414
- BRDR, 414
- BWT, 396, 397, 399–405
- BWTDONE, 374, 388, 399, 401, 412
- BWTGO, 396, 397, 399, 402
- BWTNOHIT, 403
- DONE, 423
- FLUSH, 395
- INT, 422
- PRBBRD, 411–414, 418
- PRBBWT, 399–401, 403
- PRBDONE, 370, 371, 373, 377, 380–382, 387, 397, 402, 405, 411, 413
- PRBINV, 390–392, 404, 405, 417, 419
- PRBNOHIT, 372, 386, 414
- PRBSHR, 380–384, 387, 388, 418, 419
- PRBWIN, 370, 371, 373, 374, 417, 418
- RDEX, 365, 367, 368, 370, 373, 374, 390–392
- RDEXR, 372
- RDIO, 415, 416
- RDS, 375, 377, 378, 380, 382, 383, 387, 388
- RDSR, 386
- RDSV, 376, 381, 384
- RDV, 366, 371
- SPCL, 423
- WBCANCEL, 366, 371, 376, 381, 384, 392
- WINV, 394
- WRSTRANS, 383, 385
- WTIO, 416

Control Link, 49

fabric switch, 119
FORD, 49
forward progress market, 122
FPM, 122

OCLA, SCB Triggering, 515

Performance Counters, 514

SCB, 503, 509
SClock, 49
Serial Control Bus (See SCB), 503, 509
Serial Link, 49
SPCL, 452
SysChain, Access to SCB registers, 513