

# External Memory Interface Handbook Volume 3: Reference Material

Updated for Intel® Quartus® Prime Design Suite: **17.0**



## Contents

---

|   |           |
|---|-----------|
| <b>1. Functional Description—UniPHY.....</b>  | <b>13</b> |
| 1.1. I/O Pads.....  | 14        |
| 1.2. Reset and Clock Generation.....  | 14        |
| 1.3. Dedicated Clock Networks.....  | 14        |
| 1.4. Address and Command Datapath.....  | 15        |
| 1.5. Write Datapath.....  | 16        |
| 1.5.1. Leveling Circuitry.....  | 17        |
| 1.6. Read Datapath.....   | 18        |
| 1.7. Sequencer.....   | 19        |
| 1.7.1. Nios II-Based Sequencer.....   | 20        |
| 1.7.2. RTL-based Sequencer.....   | 25        |
| 1.8. Shadow Registers.....  | 26        |
| 1.8.1. Shadow Registers Operation.....  | 27        |
| 1.9. UniPHY Interfaces.....   | 28        |
| 1.9.1. The DLL and PLL Sharing Interface.....   | 29        |
| 1.9.2. The OCT Sharing Interface.....   | 30        |
| 1.10. UniPHY Signals.....   | 32        |
| 1.11. PHY-to-Controller Interfaces.....   | 35        |
| 1.12. Using a Custom Controller.....  | 39        |
| 1.13. AFI 3.0 Specification.....  | 40        |
| 1.13.1. Bus Width and AFI Ratio.....  | 40        |
| 1.13.2. AFI Parameters.....   | 41        |
| 1.13.3. AFI Signals.....  | 42        |
| 1.14. Register Maps.....  | 46        |
| 1.14.1. UniPHY Register Map.....  | 47        |
| 1.14.2. Controller Register Map.....  | 49        |
| 1.15. Ping Pong PHY.....  | 49        |
| 1.15.1. Ping Pong PHY Feature Description.....  | 49        |
| 1.15.2. Ping Pong PHY Architecture.....   | 51        |
| 1.15.3. Ping Pong PHY Operation.....  | 52        |
| 1.16. Efficiency Monitor and Protocol Checker.....  | 52        |
| 1.16.1. Efficiency Monitor.....   | 53        |
| 1.16.2. Protocol Checker.....   | 53        |
| 1.16.3. Read Latency Counter.....   | 53        |
| 1.16.4. Using the Efficiency Monitor and Protocol Checker.....                              | 53        |
| 1.16.5. Avalon CSR Slave and JTAG Memory Map.....   | 54        |
| 1.17. UniPHY Calibration Stages.....  | 55        |
| 1.17.1. Calibration Overview.....   | 55        |
| 1.17.2. Calibration Stages.....   | 56        |
| 1.17.3. Memory Initialization.....  | 57        |
| 1.17.4. Stage 1: Read Calibration Part One—DQS Enable Calibration and DQ/DQS Centering..... | 57        |
| 1.17.5. Stage 2: Write Calibration Part One.....  | 62        |
| 1.17.6. Stage 3: Write Calibration Part Two—DQ/DQS Centering.....                           | 63        |
| 1.17.7. Stage 4: Read Calibration Part Two—Read Latency Minimization.....                   | 63        |
| 1.17.8. Calibration Signals.....  | 64        |
| 1.17.9. Calibration Time.....   | 64        |



|   |           |
|---|-----------|
| 1.18. Document Revision History.....  | 64        |
| <b>2. Functional Description—Intel Stratix® 10 EMIF IP.....</b>                   | <b>67</b> |
| 2.1. Stratix 10 Supported Memory Protocols.....                                   | 67        |
| 2.2. Stratix 10 EMIF IP Support for 3DS/TSV DDR4 Devices.....                     | 68        |
| 2.3. Migrating to Stratix 10 from Previous Device Families.....                   | 68        |
| 2.4. Stratix 10 EMIF Architecture: Introduction.....                              | 68        |
| 2.4.1. Stratix 10 EMIF Architecture: I/O Subsystem.....                           | 69        |
| 2.4.2. Stratix 10 EMIF Architecture: I/O Column.....                              | 70        |
| 2.4.3. Stratix 10 EMIF Architecture: I/O SSM.....                                 | 71        |
| 2.4.4. Stratix 10 EMIF Architecture: I/O Bank.....                                | 71        |
| 2.4.5. Stratix 10 EMIF Architecture: I/O Lane.....                                | 72        |
| 2.4.6. Stratix 10 EMIF Architecture: Input DQS Clock Tree.....                    | 74        |
| 2.4.7. Stratix 10 EMIF Architecture: PHY Clock Tree.....                          | 75        |
| 2.4.8. Stratix 10 EMIF Architecture: PLL Reference Clock Networks.....            | 75        |
| 2.4.9. Stratix 10 EMIF Architecture: Clock Phase Alignment.....                   | 76        |
| 2.5. Hardware Resource Sharing Among Multiple Stratix 10 EMIFs.....               | 77        |
| 2.5.1. I/O SSM Sharing.....   | 77        |
| 2.5.2. I/O Bank Sharing.....  | 78        |
| 2.5.3. PLL Reference Clock Sharing.....   | 79        |
| 2.5.4. Core Clock Network Sharing.....  | 80        |
| 2.6. Stratix 10 EMIF IP Component.....  | 81        |
| 2.6.1. Instantiating Your Stratix 10 EMIF IP in a Qsys Project.....               | 81        |
| 2.6.2. File Sets.....   | 84        |
| 2.6.3. Customized readme.txt File.....  | 84        |
| 2.6.4. Clock Domains.....   | 85        |
| 2.6.5. ECC in Stratix 10 EMIF IP.....   | 85        |
| 2.7. Examples of External Memory Interface Implementations for DDR4.....          | 87        |
| 2.8. Stratix 10 EMIF Sequencer.....   | 92        |
| 2.8.1. Stratix 10 EMIF DQS Tracking.....  | 93        |
| 2.9. Stratix 10 EMIF Calibration.....   | 93        |
| 2.9.1. Stratix 10 Calibration Stages .....  | 94        |
| 2.9.2. Stratix 10 Calibration Stages Descriptions.....                            | 94        |
| 2.9.3. Stratix 10 Calibration Algorithms.....                                     | 95        |
| 2.9.4. Stratix 10 Calibration Flowchart.....                                      | 98        |
| 2.10. Stratix 10 EMIF and SmartVID.....   | 99        |
| 2.11. Stratix 10 Hard Memory Controller Rate Conversion Feature.....              | 99        |
| 2.12. Differences Between User-Requested Reset in Stratix 10 versus Arria 10..... | 100       |
| 2.12.1. Method for Initiating a User-requested Reset.....                         | 101       |
| 2.13. Compiling Stratix 10 EMIF IP with the Quartus Prime Software.....           | 102       |
| 2.13.1. Instantiating the Stratix 10 EMIF IP.....                                 | 102       |
| 2.13.2. Setting I/O Assignments in Stratix 10 EMIF IP.....                        | 103       |
| 2.14. Debugging Stratix 10 EMIF IP.....   | 103       |
| 2.14.1. External Memory Interface Debug Toolkit.....                              | 103       |
| 2.14.2. On-Chip Debug for Stratix 10.....   | 104       |
| 2.14.3. Configuring Your EMIF IP for Use with the Debug Toolkit.....              | 104       |
| 2.14.4. Stratix 10 EMIF Debugging Examples.....                                   | 105       |
| 2.15. Stratix 10 EMIF for Hard Processor Subsystem.....                           | 107       |
| 2.15.1. Restrictions on I/O Bank Usage for Stratix 10 EMIF IP with HPS.....       | 108       |
| 2.16. Stratix 10 EMIF Ping Pong PHY.....  | 111       |
| 2.16.1. Stratix 10 Ping Pong PHY Feature Description.....                         | 111       |



|  |            |
|--|------------|
| 2.16.2. Stratix 10 Ping Pong PHY Architecture.....                           | 112        |
| 2.16.3. Stratix 10 Ping Pong PHY Limitations.....                            | 114        |
| 2.16.4. Stratix 10 Ping Pong PHY Calibration.....                            | 116        |
| 2.16.5. Using the Ping Pong PHY.....   | 117        |
| 2.16.6. Ping Pong PHY Simulation Example Design.....                         | 117        |
| 2.17. AFI 4.0 Specification.....   | 117        |
| 2.17.1. Bus Width and AFI Ratio.....   | 118        |
| 2.17.2. AFI Parameters.....  | 118        |
| 2.17.3. AFI Signals.....   | 120        |
| 2.17.4. AFI 4.0 Timing Diagrams.....   | 125        |
| 2.18. Stratix 10 Resource Utilization.....                                   | 139        |
| 2.18.1. QDR-IV Resource Utilization in Stratix 10 Devices.....               | 139        |
| 2.19. Stratix 10 EMIF Latency.....   | 139        |
| 2.20. Integrating a Custom Controller with the Hard PHY.....                 | 140        |
| 2.21. Document Revision History.....   | 140        |
| <b>3. Functional Description—Intel Arria 10 EMIF IP.....</b>                 | <b>141</b> |
| 3.1. Supported Memory Protocols.....   | 141        |
| 3.2. Key Differences Compared to UniPHY IP and Previous Device Families..... | 142        |
| 3.3. Migrating from Previous Device Families.....                            | 143        |
| 3.4. Arria 10 EMIF Architecture: Introduction.....                           | 143        |
| 3.4.1. Arria 10 EMIF Architecture: I/O Subsystem.....                        | 144        |
| 3.4.2. Arria 10 EMIF Architecture: I/O Column.....                           | 144        |
| 3.4.3. Arria 10 EMIF Architecture: I/O AUX.....                              | 145        |
| 3.4.4. Arria 10 EMIF Architecture: I/O Bank.....                             | 145        |
| 3.4.5. Arria 10 EMIF Architecture: I/O Lane.....                             | 149        |
| 3.4.6. Arria 10 EMIF Architecture: Input DQS Clock Tree.....                 | 151        |
| 3.4.7. Arria 10 EMIF Architecture: PHY Clock Tree.....                       | 152        |
| 3.4.8. Arria 10 EMIF Architecture: PLL Reference Clock Networks.....         | 152        |
| 3.4.9. Arria 10 EMIF Architecture: Clock Phase Alignment.....                | 153        |
| 3.5. Hardware Resource Sharing Among Multiple EMIFs.....                     | 154        |
| 3.5.1. I/O Aux Sharing.....  | 154        |
| 3.5.2. I/O Bank Sharing.....   | 155        |
| 3.5.3. PLL Reference Clock Sharing.....                                      | 156        |
| 3.5.4. Core Clock Network Sharing.....                                       | 157        |
| 3.6. Arria 10 EMIF IP Component.....   | 158        |
| 3.6.1. Instantiating Your Arria 10 EMIF IP in a Qsys Project.....            | 158        |
| 3.6.2. File Sets.....  | 161        |
| 3.6.3. Customized readme.txt File.....                                       | 161        |
| 3.6.4. Clock Domains.....  | 162        |
| 3.6.5. ECC in Arria 10 EMIF IP.....  | 162        |
| 3.7. Examples of External Memory Interface Implementations for DDR4.....     | 164        |
| 3.8. Arria 10 EMIF Sequencer.....  | 169        |
| 3.8.1. Arria 10 EMIF DQS Tracking.....                                       | 170        |
| 3.9. Arria 10 EMIF Calibration.....  | 170        |
| 3.9.1. Calibration Stages .....  | 170        |
| 3.9.2. Calibration Stages Descriptions.....                                  | 171        |
| 3.9.3. Calibration Algorithms.....   | 172        |
| 3.9.4. Calibration Flowchart.....  | 175        |
| 3.9.5. Periodic OCT Recalibration.....                                       | 176        |
| 3.10. Back-to-Back User-Controlled Refresh Usage in Arria 10.....            | 178        |



|  |     |
|--|-----|
| 3.11. Arria 10 EMIF and SmartVID.....                                      | 178 |
| 3.12. Hard Memory Controller Rate Conversion Feature.....                  | 179 |
| 3.13. Back-to-Back User-Controlled Refresh for Hard Memory Controller..... | 180 |
| 3.14. Compiling Arria 10 EMIF IP with the Quartus Prime Software.....      | 180 |
| 3.14.1. Instantiating the Arria 10 EMIF IP.....                            | 181 |
| 3.14.2. Setting I/O Assignments in Arria 10 EMIF IP.....                   | 181 |
| 3.15. Debugging Arria 10 EMIF IP.....                                      | 181 |
| 3.15.1. External Memory Interface Debug Toolkit.....                       | 182 |
| 3.15.2. On-Chip Debug for Arria 10.....                                    | 182 |
| 3.15.3. Configuring Your EMIF IP for Use with the Debug Toolkit.....       | 182 |
| 3.15.4. Arria 10 EMIF Debugging Examples.....                              | 183 |
| 3.16. Arria 10 EMIF for Hard Processor Subsystem.....                      | 185 |
| 3.16.1. Restrictions on I/O Bank Usage for Arria 10 EMIF IP with HPS.....  | 186 |
| 3.16.2. Using the EMIF Debug Toolkit with Arria 10 HPS Interfaces.....     | 188 |
| 3.17. Arria 10 EMIF Ping Pong PHY.....                                     | 188 |
| 3.17.1. Ping Pong PHY Feature Description.....                             | 188 |
| 3.17.2. Ping Pong PHY Architecture.....                                    | 190 |
| 3.17.3. Ping Pong PHY Limitations.....                                     | 191 |
| 3.17.4. Ping Pong PHY Calibration.....                                     | 192 |
| 3.17.5. Using the Ping Pong PHY.....                                       | 193 |
| 3.17.6. Ping Pong PHY Simulation Example Design.....                       | 193 |
| 3.18. AFI 4.0 Specification.....   | 194 |
| 3.18.1. Bus Width and AFI Ratio.....                                       | 194 |
| 3.18.2. AFI Parameters.....  | 195 |
| 3.18.3. AFI Signals.....   | 196 |
| 3.18.4. AFI 4.0 Timing Diagrams.....                                       | 201 |
| 3.19. Resource Utilization.....  | 215 |
| 3.19.1. QDR-IV Resource Utilization in Arria 10 Devices.....               | 215 |
| 3.20. Arria 10 EMIF Latency.....   | 215 |
| 3.21. Arria 10 EMIF Calibration Times.....                                 | 216 |
| 3.22. Integrating a Custom Controller with the Hard PHY.....               | 217 |
| 3.23. Memory Mapped Register (MMR) Tables.....                             | 217 |
| 3.23.1. ctrlcfg0: Controller Configuration.....                            | 219 |
| 3.23.2. ctrlcfg1: Controller Configuration.....                            | 220 |
| 3.23.3. ctrlcfg2: Controller Configuration.....                            | 221 |
| 3.23.4. ctrlcfg3: Controller Configuration.....                            | 222 |
| 3.23.5. ctrlcfg4: Controller Configuration.....                            | 223 |
| 3.23.6. ctrlcfg5: Controller Configuration.....                            | 224 |
| 3.23.7. ctrlcfg6: Controller Configuration.....                            | 225 |
| 3.23.8. ctrlcfg7: Controller Configuration.....                            | 225 |
| 3.23.9. ctrlcfg8: Controller Configuration.....                            | 225 |
| 3.23.10. ctrlcfg9: Controller Configuration.....                           | 225 |
| 3.23.11. dramtiming0: Timing Parameters.....                               | 226 |
| 3.23.12. dramodt0: On-Die Termination Parameters.....                      | 226 |
| 3.23.13. dramodt1: On-Die Termination Parameters.....                      | 226 |
| 3.23.14. sbcfg0: Sideband Configuration.....                               | 227 |
| 3.23.15. sbcfg1: Sideband Configuration.....                               | 227 |
| 3.23.16. sbcfg2: Sideband Configuration.....                               | 227 |
| 3.23.17. sbcfg3: Sideband Configuration.....                               | 228 |
| 3.23.18. sbcfg4: Sideband Configuration.....                               | 228 |
| 3.23.19. sbcfg5: Sideband Configuration.....                               | 228 |



|  |            |
|--|------------|
| 3.23.20. sbcfg6: Sideband Configuration.....                         | 229        |
| 3.23.21. sbcfg7: Sideband Configuration.....                         | 229        |
| 3.23.22. sbcfg8: Sideband Configuration.....                         | 229        |
| 3.23.23. sbcfg9: Sideband Configuration.....                         | 229        |
| 3.23.24. caltiming0: Command/Address/Latency Parameters.....         | 230        |
| 3.23.25. caltiming1: Command/Address/Latency Parameters.....         | 230        |
| 3.23.26. caltiming2: Command/Address/Latency Parameters.....         | 230        |
| 3.23.27. caltiming3: Command/Address/Latency Parameters.....         | 231        |
| 3.23.28. caltiming4: Command/Address/Latency Parameters.....         | 231        |
| 3.23.29. caltiming5: Command/Address/Latency Parameters.....         | 231        |
| 3.23.30. caltiming6: Command/Address/Latency Parameters.....         | 232        |
| 3.23.31. caltiming7: Command/Address/Latency Parameters.....         | 232        |
| 3.23.32. caltiming8: Command/Address/Latency Parameters.....         | 232        |
| 3.23.33. caltiming9: Command/Address/Latency Parameters.....         | 233        |
| 3.23.34. caltiming10: Command/Address/Latency Parameters.....        | 233        |
| 3.23.35. dramaddrw: Row/Column/Bank Address Width Configuration..... | 233        |
| 3.23.36. sideband0: Sideband.....                                    | 234        |
| 3.23.37. sideband1: Sideband.....                                    | 234        |
| 3.23.38. sideband2: Sideband.....                                    | 234        |
| 3.23.39. sideband3: Sideband.....                                    | 234        |
| 3.23.40. sideband4: Sideband.....                                    | 234        |
| 3.23.41. sideband5: Sideband.....                                    | 235        |
| 3.23.42. sideband6: Sideband.....                                    | 235        |
| 3.23.43. sideband7: Sideband.....                                    | 235        |
| 3.23.44. sideband8: Sideband.....                                    | 235        |
| 3.23.45. sideband9: Sideband.....                                    | 235        |
| 3.23.46. sideband10: Sideband.....                                   | 235        |
| 3.23.47. sideband11: Sideband.....                                   | 236        |
| 3.23.48. sideband12: Sideband.....                                   | 236        |
| 3.23.49. sideband13: Sideband.....                                   | 236        |
| 3.23.50. sideband14: Sideband.....                                   | 236        |
| 3.23.51. sideband15: Sideband.....                                   | 237        |
| 3.23.52. dramsts: Calibration Status.....                            | 237        |
| 3.23.53. ecc1: ECC General Configuration.....                        | 237        |
| 3.23.54. ecc2: Width Configuration.....                              | 238        |
| 3.23.55. ecc3: ECC Error and Interrupt Configuration.....            | 238        |
| 3.23.56. ecc4: Status and Error Information.....                     | 239        |
| 3.23.57. ecc5: Address of Most Recent SBE/DBE.....                   | 239        |
| 3.23.58. ecc6: Address of Most Recent Correct Command Dropped.....   | 240        |
| 3.24. Document Revision History.....                                 | 240        |
| <b>4. Functional Description—Intel MAX® 10 EMIF IP.....</b>          | <b>242</b> |
| 4.1. MAX 10 EMIF Overview.....                                       | 242        |
| 4.2. External Memory Protocol Support.....                           | 243        |
| 4.3. MAX 10 Memory Controller.....                                   | 243        |
| 4.4. MAX 10 Low Power Feature.....                                   | 243        |
| 4.5. MAX 10 Memory PHY.....  | 244        |
| 4.5.1. Supported Topologies.....                                     | 244        |
| 4.5.2. Read Datapath.....  | 244        |
| 4.5.3. Write Datapath.....   | 245        |
| 4.5.4. Address and Command Datapath.....                             | 247        |



|   |            |
|---|------------|
| 4.5.5. Sequencer.....   | 247        |
| 4.6. Calibration.....   | 250        |
| 4.6.1. Read Calibration.....  | 250        |
| 4.6.2. Write Calibration.....   | 250        |
| 4.7. Sequencer Debug Information.....   | 251        |
| 4.8. Register Maps.....   | 252        |
| 4.9. Document Revision History.....   | 252        |
| <b>6. Functional Description—Hard Memory Interface.....</b>                     | <b>253</b> |
| 6.1. Multi-Port Front End (MPFE).....   | 254        |
| 6.2. Multi-port Scheduling.....   | 255        |
| 6.2.1. Port Scheduling.....   | 255        |
| 6.2.2. DRAM Burst Scheduling.....   | 255        |
| 6.2.3. DRAM Power Saving Modes.....   | 256        |
| 6.3. MPFE Signal Descriptions.....  | 256        |
| 6.4. Hard Memory Controller.....  | 259        |
| 6.4.1. Clocking.....  | 260        |
| 6.4.2. Reset.....   | 260        |
| 6.4.3. DRAM Interface.....  | 260        |
| 6.4.4. ECC.....   | 260        |
| 6.4.5. Bonding of Memory Controllers.....                                       | 261        |
| 6.5. Hard PHY.....  | 263        |
| 6.5.1. Interconnections.....  | 263        |
| 6.5.2. Clock Domains.....   | 263        |
| 6.5.3. Hard Sequencer.....  | 263        |
| 6.5.4. MPFE Setup Guidelines.....   | 264        |
| 6.5.5. Soft Memory Interface to Hard Memory Interface Migration Guidelines..... | 265        |
| 6.5.6. Bonding Interface Guidelines.....  | 266        |
| 6.6. Document Revision History.....   | 266        |
| <b>7. Functional Description—HPS Memory Controller.....</b>                     | <b>268</b> |
| 7.1. Features of the SDRAM Controller Subsystem.....                            | 268        |
| 7.2. SDRAM Controller Subsystem Block Diagram.....                              | 269        |
| 7.3. SDRAM Controller Memory Options.....                                       | 270        |
| 7.4. SDRAM Controller Subsystem Interfaces.....                                 | 271        |
| 7.4.1. MPU Subsystem Interface.....   | 271        |
| 7.4.2. L3 Interconnect Interface.....   | 271        |
| 7.4.3. CSR Interface.....   | 271        |
| 7.4.4. FPGA-to-HPS SDRAM Interface.....   | 271        |
| 7.5. Memory Controller Architecture.....  | 272        |
| 7.5.1. Multi-Port Front End.....  | 273        |
| 7.5.2. Single-Port Controller.....  | 274        |
| 7.6. Functional Description of the SDRAM Controller Subsystem.....              | 276        |
| 7.6.1. MPFE Operation Ordering.....   | 276        |
| 7.6.2. MPFE Multi-Port Arbitration.....   | 276        |
| 7.6.3. MPFE SDRAM Burst Scheduling.....   | 280        |
| 7.6.4. Single-Port Controller Operation.....                                    | 281        |
| 7.7. SDRAM Power Management.....  | 291        |
| 7.8. DDR PHY.....   | 292        |
| 7.9. Clocks.....  | 292        |
| 7.10. Resets.....   | 292        |



|  |            |
|--|------------|
| 7.10.1. Taking the SDRAM Controller Subsystem Out of Reset .....         | 293        |
| 7.11. Port Mappings.....   | 293        |
| 7.12. Initialization.....  | 293        |
| 7.12.1. FPGA-to-SDRAM Protocol Details.....                              | 294        |
| 7.13. SDRAM Controller Subsystem Programming Model.....                  | 297        |
| 7.13.1. HPS Memory Interface Architecture.....                           | 298        |
| 7.13.2. HPS Memory Interface Configuration.....                          | 298        |
| 7.13.3. HPS Memory Interface Simulation.....                             | 298        |
| 7.13.4. Generating a Preloader Image for HPS with EMIF.....              | 299        |
| 7.14. Debugging HPS SDRAM in the Preloader.....                          | 300        |
| 7.14.1. Enabling UART or Semihosting Printout.....                       | 300        |
| 7.14.2. Enabling Simple Memory Test.....                                 | 301        |
| 7.14.3. Enabling the Debug Report.....                                   | 303        |
| 7.14.4. Writing a Predefined Data Pattern to SDRAM in the Preloader..... | 305        |
| 7.15. SDRAM Controller Address Map and Register Definitions.....         | 307        |
| 7.15.1. SDRAM Controller Address Map.....                                | 307        |
| 7.16. Document Revision History.....                                     | 335        |
| <b>8. Functional Description—HPC II Controller.....</b>                  | <b>337</b> |
| 8.1. HPC II Memory Interface Architecture.....                           | 337        |
| 8.2. HPC II Memory Controller Architecture.....                          | 338        |
| 8.2.1. Backpressure Support.....   | 340        |
| 8.2.2. Command Generator.....  | 341        |
| 8.2.3. Timing Bank Pool.....   | 341        |
| 8.2.4. Arbiter.....  | 341        |
| 8.2.5. Rank Timer.....   | 341        |
| 8.2.6. Read Data Buffer and Write Data Buffer.....                       | 341        |
| 8.2.7. ECC Block.....  | 342        |
| 8.2.8. AFI and CSR Interfaces.....                                       | 342        |
| 8.3. HPC II Controller Features.....                                     | 342        |
| 8.3.1. Data Reordering.....  | 342        |
| 8.3.2. Pre-emptive Bank Management.....                                  | 342        |
| 8.3.3. Quasi-1T and Quasi-2T.....  | 343        |
| 8.3.4. User Autoprecharge Commands.....                                  | 343        |
| 8.3.5. Address and Command Decoding Logic.....                           | 343        |
| 8.3.6. Low-Power Logic.....  | 343        |
| 8.3.7. ODT Generation Logic.....   | 344        |
| 8.3.8. Burst Merging.....  | 346        |
| 8.3.9. ECC.....  | 346        |
| 8.4. External Interfaces.....  | 349        |
| 8.4.1. Clock and Reset Interface.....                                    | 349        |
| 8.4.2. Avalon-ST Data Slave Interface.....                               | 349        |
| 8.4.3. AXI Data Slave Interface.....                                     | 349        |
| 8.4.4. Controller-PHY Interface.....                                     | 355        |
| 8.4.5. Memory Side-Band Signals.....                                     | 355        |
| 8.4.6. Controller External Interfaces.....                               | 356        |
| 8.5. Top-Level Signals Description.....                                  | 357        |
| 8.5.1. Clock and Reset Signals.....                                      | 357        |
| 8.5.2. Local Interface Signals.....                                      | 358        |
| 8.5.3. Controller Interface Signals.....                                 | 362        |
| 8.5.4. CSR Interface Signals.....  | 363        |



|   |            |
|---|------------|
| 8.5.5. Soft Controller Register Map.....                              | 364        |
| 8.5.6. Hard Controller Register Map.....                              | 368        |
| 8.6. Sequence of Operations.....                                      | 371        |
| 8.7. Document Revision History.....                                   | 373        |
| <b>9. Functional Description—QDR II Controller.....</b>               | <b>375</b> |
| 9.1. Block Description.....   | 375        |
| 9.1.1. Avalon-MM Slave Read and Write Interfaces.....                 | 375        |
| 9.1.2. Command Issuing FSM.....                                       | 376        |
| 9.1.3. AFI.....   | 376        |
| 9.2. Avalon-MM and Memory Data Width.....                             | 376        |
| 9.3. Signal Descriptions.....   | 377        |
| 9.4. Document Revision History.....                                   | 378        |
| <b>10. Functional Description—QDR-IV Controller.....</b>              | <b>379</b> |
| 10.1. Block Description.....  | 379        |
| 10.1.1. Avalon-MM Slave Read and Write Interfaces.....                | 379        |
| 10.1.2. AFI.....  | 380        |
| 10.2. Avalon-MM and Memory Data Width.....                            | 380        |
| 10.3. Signal Descriptions.....  | 381        |
| 10.4. Document Revision History.....                                  | 382        |
| <b>11. Functional Description—RLDRAM II Controller.....</b>           | <b>383</b> |
| 11.1. Block Description.....  | 383        |
| 11.1.1. Avalon-MM Slave Interface.....                                | 383        |
| 11.1.2. Write Data FIFO Buffer.....                                   | 384        |
| 11.1.3. Command Issuing FSM.....                                      | 384        |
| 11.1.4. Refresh Timer.....  | 384        |
| 11.1.5. Timer Module.....   | 384        |
| 11.1.6. AFI.....  | 384        |
| 11.2. User-Controlled Features.....                                   | 385        |
| 11.2.1. Error Detection Parity.....                                   | 385        |
| 11.2.2. User-Controlled Refresh.....                                  | 385        |
| 11.3. Avalon-MM and Memory Data Width.....                            | 385        |
| 11.4. Signal Descriptions.....  | 385        |
| 11.5. Document Revision History.....                                  | 386        |
| <b>12. Functional Description—RLDRAM 3 PHY-Only IP.....</b>           | <b>388</b> |
| 12.1. Block Description.....  | 388        |
| 12.2. Features.....   | 388        |
| 12.3. RLDRAM 3 AFI Protocol.....                                      | 389        |
| 12.4. RLDRAM 3 Controller with Arria 10 EMIF Interfaces.....          | 390        |
| 12.5. Document Revision History.....                                  | 391        |
| <b>13. Functional Description—Example Designs.....</b>                | <b>392</b> |
| 13.1. Arria 10 EMIF IP Example Designs Quick Start Guide.....         | 392        |
| 13.1.1. Typical Example Design Workflow.....                          | 392        |
| 13.1.2. Example Designs Interface Tab.....                            | 393        |
| 13.1.3. Development Kit Preset Workflow.....                          | 395        |
| 13.1.4. Compiling and Simulating the Design.....                      | 396        |
| 13.1.5. Compiling and Testing the Design in Hardware.....             | 397        |
| 13.2. Testing the EMIF Interface Using the Traffic Generator 2.0..... | 398        |



|   |            |
|---|------------|
| 13.2.1. Configurable Traffic Generator 2.0 Configuration Options.....   | 399        |
| 13.2.2. Performing Your Own Tests Using Traffic Generator 2.0.....  | 404        |
| 13.2.3. Signal Splitter Component .....   | 405        |
| 13.3. UniPHY-Based Example Designs.....   | 406        |
| 13.3.1. Synthesis Example Design.....   | 406        |
| 13.3.2. Simulation Example Design.....  | 407        |
| 13.3.3. Traffic Generator and BIST Engine.....  | 409        |
| 13.3.4. Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys.....                              | 412        |
| 13.4. Document Revision History.....  | 414        |
| <b>14. Introduction to UniPHY IP.....</b>   | <b>416</b> |
| 14.1. Release Information.....  | 416        |
| 14.2. Device Support Levels.....  | 417        |
| 14.3. Device Family and Protocol Support.....   | 417        |
| 14.4. UniPHY-Based External Memory Interface Features.....  | 418        |
| 14.5. System Requirements.....  | 419        |
| 14.6. Intel FPGA IP Core Verification.....  | 419        |
| 14.7. Resource Utilization.....   | 419        |
| 14.7.1. DDR2, DDR3, and LPDDR2 Resource Utilization in Arria V Devices.....   | 420        |
| 14.7.2. DDR2 and DDR3 Resource Utilization in Arria II GZ Devices.....  | 421        |
| 14.7.3. DDR2 and DDR3 Resource Utilization in Stratix III Devices.....  | 422        |
| 14.7.4. DDR2 and DDR3 Resource Utilization in Stratix IV Devices.....   | 423        |
| 14.7.5. DDR2 and DDR3 Resource Utilization in Arria V GZ and Stratix V Devices.....   | 424        |
| 14.7.6. QDR II and QDR II+ Resource Utilization in Arria V Devices.....   | 426        |
| 14.7.7. QDR II and QDR II+ Resource Utilization in Arria II GX Devices.....   | 427        |
| 14.7.8. QDR II and QDR II+ Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices..... | 427        |
| 14.7.9. RLDRAM II Resource Utilization in Arria V Devices.....  | 427        |
| 14.7.10. RLDRAM II Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices.....         | 428        |
| 14.8. Document Revision History.....  | 428        |
| <b>15. Latency for UniPHY IP.....</b>   | <b>430</b> |
| 15.1. DDR2 SDRAM LATENCY.....   | 430        |
| 15.2. DDR3 SDRAM LATENCY.....   | 431        |
| 15.3. LPDDR2 SDRAM LATENCY.....   | 431        |
| 15.4. QDR II and QDR II+ SRAM Latency.....  | 432        |
| 15.5. RLDRAM II Latency.....  | 432        |
| 15.6. RLDRAM 3 Latency.....   | 433        |
| 15.7. Variable Controller Latency.....  | 433        |
| 15.8. Document Revision History.....  | 433        |
| <b>16. Timing Diagrams for UniPHY IP.....</b>   | <b>435</b> |
| 16.1. DDR2 Timing Diagrams.....   | 435        |
| 16.2. DDR3 Timing Diagrams.....   | 440        |
| 16.3. QDR II and QDR II+ Timing Diagrams.....   | 448        |
| 16.4. RLDRAM II Timing Diagrams.....  | 452        |
| 16.5. LPDDR2 Timing Diagrams.....   | 458        |
| 16.6. RLDRAM 3 Timing Diagrams.....   | 465        |
| 16.7. Document Revision History.....  | 469        |



|  |            |
|--|------------|
| <b>17. External Memory Interface Debug Toolkit.....</b>  | <b>471</b> |
| 17.1. User Interface.....  | 471        |
| 17.1.1. Communication.....   | 471        |
| 17.1.2. Calibration and Report Generation.....   | 472        |
| 17.2. Setup and Use.....   | 472        |
| 17.2.1. General Workflow.....  | 473        |
| 17.2.2. Linking the Project to a Device.....   | 473        |
| 17.2.3. Establishing Communication to Connections.....   | 474        |
| 17.2.4. Selecting an Active Interface.....   | 474        |
| 17.2.5. Reports.....   | 475        |
| 17.3. Operational Considerations.....  | 476        |
| 17.4. Troubleshooting.....   | 477        |
| 17.5. Debug Report for Arria V and Cyclone V SoC Devices.....  | 478        |
| 17.5.1. Enabling the Debug Report for Arria V and Cyclone V SoC Devices.....                           | 478        |
| 17.5.2. Determining the Failing Calibration Stage for a Cyclone V or Arria V HPS SDRAM Controller..... | 478        |
| 17.6. On-Chip Debug Port for UniPHY-based EMIF IP.....   | 479        |
| 17.6.1. Access Protocol.....   | 480        |
| 17.6.2. Command Codes Reference.....   | 481        |
| 17.6.3. Header Files.....  | 482        |
| 17.6.4. Generating IP With the Debug Port.....   | 482        |
| 17.6.5. Example C Code for Accessing Debug Data.....   | 483        |
| 17.7. On-Chip Debug Port for Arria 10 EMIF IP.....   | 484        |
| 17.7.1. Access Protocol.....   | 485        |
| 17.7.2. EMIF On-Chip Debug Port.....   | 486        |
| 17.7.3. On-Die Termination Calibration .....   | 487        |
| 17.7.4. Eye Diagram .....  | 487        |
| 17.8. Driver Margining for Arria 10 EMIF IP.....   | 487        |
| 17.8.1. Determining Margin.....  | 488        |
| 17.9. Read Setting and Apply Setting Commands for Arria 10 EMIF IP.....                                | 488        |
| 17.9.1. Reading or Applying Calibration Settings.....  | 488        |
| 17.10. Traffic Generator 2.0.....  | 489        |
| 17.10.1. Configuring the Traffic Generator 2.0.....  | 489        |
| 17.10.2. Running the Traffic Generator 2.0.....  | 491        |
| 17.10.3. Understanding the Custom Traffic Generator User Interface.....                                | 491        |
| 17.10.4. Applying the Traffic Generator 2.0.....   | 496        |
| 17.11. The Traffic Generator 2.0 Report.....   | 498        |
| 17.12. Example Tcl Script for Running the EMIF Debug Toolkit.....                                      | 499        |
| 17.13. Calibration Adjustment Delay Step Sizes for Arria 10 Devices.....                               | 500        |
| 17.13.1. Addressing.....   | 500        |
| 17.13.2. Output and Strobe Enable Minimum and Maximum Phase Settings.....                              | 503        |
| 17.14. Using the EMIF Debug Toolkit with Arria 10 HPS Interfaces.....                                  | 503        |
| 17.15. Document Revision History.....  | 504        |
| <b>18. Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers.....</b>                 | <b>506</b> |
| 18.1. Generating Equivalent Design.....  | 506        |
| 18.2. Replacing the ALTMEMPHY Datapath with UniPHY Datapath.....                                       | 507        |
| 18.3. Resolving Port Name Differences.....   | 507        |
| 18.4. Creating OCT Signals.....  | 509        |
| 18.5. Running Pin Assignments Script.....  | 509        |
| 18.6. Removing Obsolete Files.....   | 509        |



**Contents**

|                                      |     |
|--------------------------------------|-----|
| 18.7. Simulating your Design.....    | 509 |
| 18.8. Document Revision History..... | 511 |

## 1. Functional Description—UniPHY

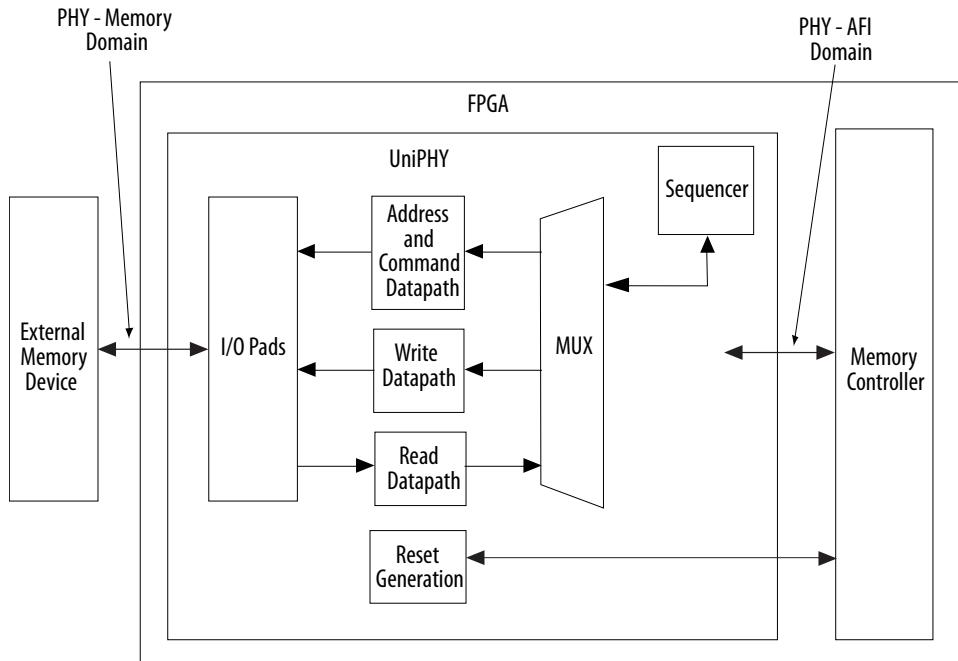
UniPHY is the physical layer of the external memory interface.

The major functional units of the UniPHY layer include the following:

- Reset and clock generation
- Address and command datapath
- Write datapath
- Read datapath
- Sequencer

The following figure shows the PHY block diagram.

**Figure 1. PHY Block Diagram**



### Related Information

- [UniPHY Interfaces](#) on page 28
- [UniPHY Signals](#) on page 32
- [PHY-to-Controller Interfaces](#) on page 35
- [Using a Custom Controller](#) on page 39



- [AFI 4.0 Specification](#) on page 117
- [Register Maps](#) on page 46
- [Ping Pong PHY](#) on page 49
- [Efficiency Monitor](#) on page 53
- [Calibration Stages](#) on page 56

## 1.1. I/O Pads

The I/O pads contain all the I/O instantiations.

## 1.2. Reset and Clock Generation

At a high level, clocks in the PHY can be classified into two domains: the PHY-memory domain and the PHY-AFI domain.

The PHY-memory domain interfaces with the external memory device and always operate at full-rate. The PHY-AFI domain interfaces with the memory controller and can be a full-rate, half-rate, or quarter-rate clock, based on the controller in use.

The number of clock domains in a memory interface can vary depending on its configuration; for example:

- At the PHY-memory boundary, separate clocks may exist to generate the memory clock signal, the output strobe, and to output write data, as well as address and command signals. These clocks include `pll_dq_write_clk`, `pll_write_clk`, `pll_mem_clk`, and `pll_addr_cmd_clk`. These clocks are phase-shifted as required to achieve the desired timing relationships between memory clock, address and command signals, output data, and output strobe.
- For quarter-rate interfaces, additional clock domains such as `pll_hr_clock` are required to convert signals between half-rate and quarter-rate.
- For high-performance memory interfaces using Arria V, Cyclone V, or Stratix V devices, additional clocks may be required to handle transfers between the device core and the I/O periphery for timing closure. For core-to-periphery transfers, the latch clock is `pll_c2p_write_clock`; for periphery-to-core transfers, it is `pll_p2c_read_clock`. These clocks are automatically phase-adjusted for timing closure during IP generation, but can be further adjusted in the parameter editor. If the phases of these clocks are zero, the Fitter may remove these clocks during optimization. Also, high-performance interfaces using a Nios II-based sequencer require two additional clocks, `pll_avl_clock` for the Nios II processor, and `pll_config_clock` for clocking the I/O scan chains during calibration.

For a complete list of clocks in your memory interface, compile your design and run the **Report Clocks** command in the TimeQuest Timing Analyzer.

## 1.3. Dedicated Clock Networks

The UniPHY layer employs three types of dedicated clock networks:



- Global clock network
- Dual-regional clock network
- PHY clock network (applicable to Arria V, Cyclone V, and Stratix V devices, and later)

The PHY clock network is a dedicated high-speed, low-skew, balanced clock tree designed for high-performance external memory interface. For device families that support the PHY clock network, UniPHY always uses the PHY clock network for all clocks at the PHY-memory boundary.

For families that do not support the PHY clock network, UniPHY uses either dual-regional or global clock networks for clocks at the PHY-memory boundary. During generation, the system selects dual-regional or global clocks automatically, depending on whether a given interface spans more than one quadrant. UniPHY does not mix the usage of dual-regional and global clock networks for clocks at the PHY-memory boundary; this ensures that timing characteristics of the various output paths are as similar as possible.

The `<variation_name>.pin_assignments.tcl` script creates the appropriate clock network type assignment. The use of the PHY clock network is specified directly in the RTL code, and does not require an assignment.

The UniPHY uses an active-low, asynchronous assert and synchronous de-assert reset scheme. The global reset signal resets the PLL in the PHY and the rest of the system is held in reset until after the PLL is locked.

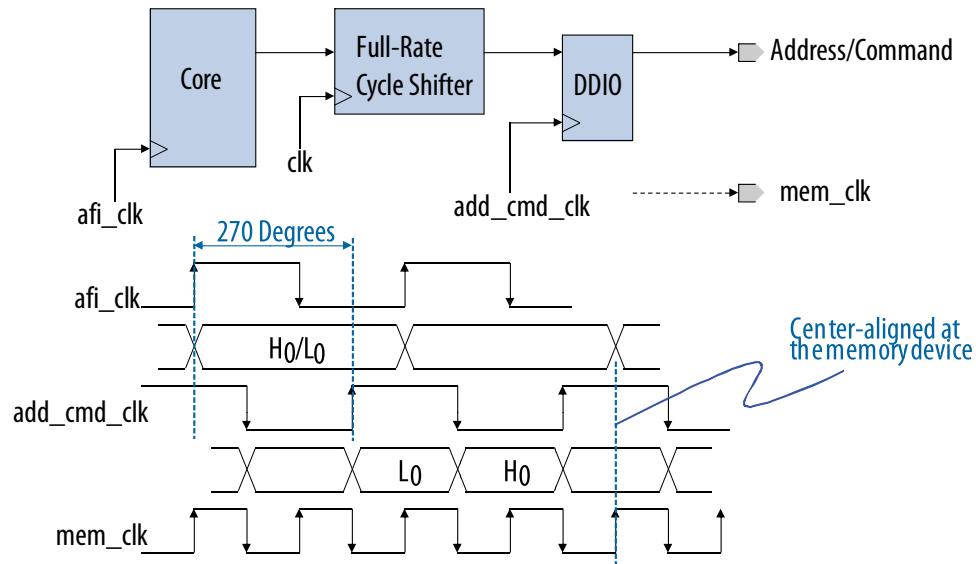
## 1.4. Address and Command Datapath

The memory controller controls the read and write addresses and commands to meet the memory specifications.

The PHY is indifferent to address or command—that is, it performs no decoding or other operations—and the circuitry is the same for both. In full-rate and half-rate interfaces, address and command is full rate, while in quarter-rate interfaces, address and command is half rate.

Address and command signals are generated in the Altera PHY interface (AFI) clock domain and sent to the memory device in the address and command clock domain. The double-data rate input/output (DDIO) stage converts the half-rate signals into full-rate signals, when the AFI clock runs at half-rate. For quarter-rate interfaces, additional DDIO stages exist to convert the address and command signals in the quarter-rate AFI clock domain to half-rate.

The address and command clock is offset with respect to the memory clock to balance the nominal setup and hold margins at the memory device (center-alignment). In the example below, this offset is 270 degrees. The Fitter can further optimize margins based on the actual delays and clock skews. In half-rate and quarter-rate designs, the full-rate cycle shifter blocks can perform a shift measured in full-rate cycles to implement the correct write latency; without this logic, the controller would only be able to implement even write latencies as it operates at half the speed. The full-rate cycle shifter is clocked by either the AFI clock or the address and command clock, depending on the PHY configuration, to maximize timing margins on the path from the AFI clock to the address and command clock.

**Figure 2. Address and Command Datapath (Half-rate example shown)**


## 1.5. Write Datapath

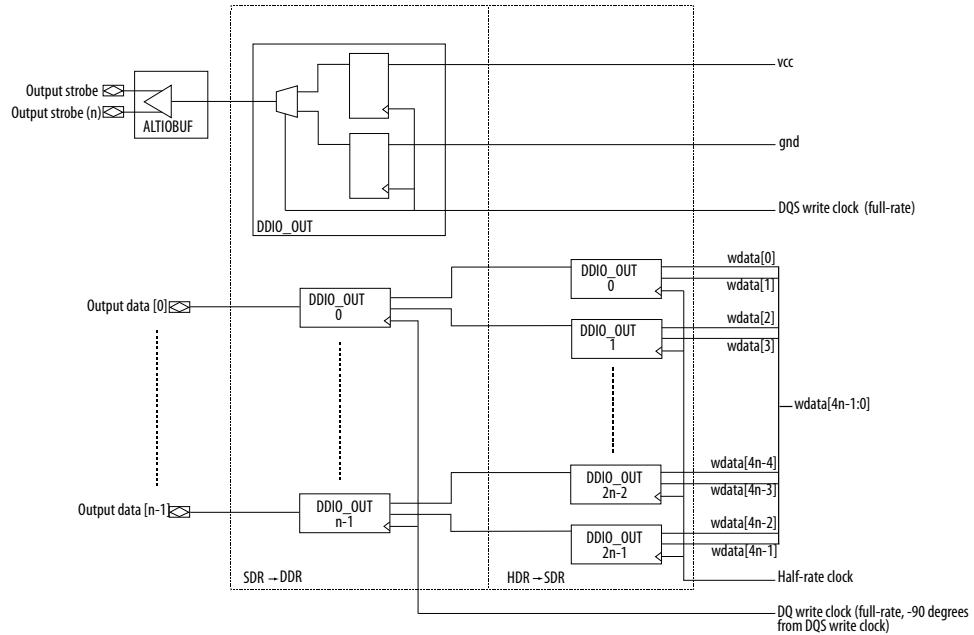
The write datapath passes write data from the memory controller to the I/O. The write data valid signal from the memory controller generates the output enable signal to control the output buffer. For memory protocols with a bidirectional data bus, it also generates the dynamic termination control signal, which selects between series (output mode) and parallel (input mode) termination.

The figure below illustrates a simplified write datapath of a typical half-rate interface. The full-rate DQS write clock is sent to a DDIO\_OUT cell. The output of DDIO\_OUT feeds an output buffer which creates a pair of pseudo differential clocks that connects to the memory. In full-rate mode, only the SDR-DDR portion of the path is used; in half-rate mode, the HDR-SDR circuitry is also required. The use of DDIO\_OUT in both the output strobe and output data generation path ensures that their timing characteristics are as similar as possible. The

`<variation_name>.pin_assignments.tcl` script automatically specifies the logic option that associates all data pins to the output strobe pin. The Fitter treats the pins as a DQS/DQ pin group.



**Figure 3. Write Datapath**



### 1.5.1. Leveling Circuitry

Leveling circuitry is dedicated I/O circuitry to provide calibration support for fly-by address and command networks. For DDR3, leveling is always invoked, whether the interface targets a DIMM or a single component. For DDR3 implementations at higher frequencies, a fly-by topology is recommended for optimal performance. For DDR2, leveling circuitry is invoked automatically for frequencies above 240 MHz; no leveling is used for frequencies below 240 MHz.

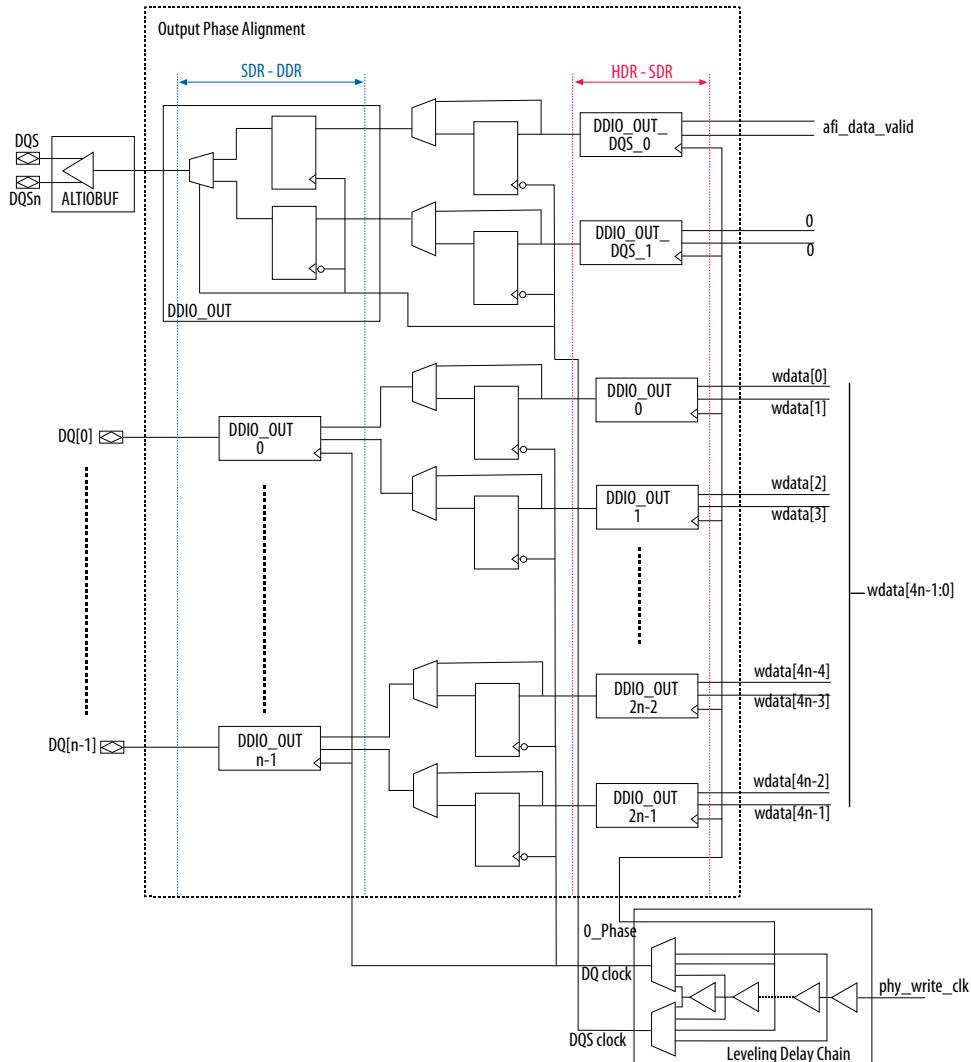
For DDR2 at frequencies below 240 MHz, you should use a tree-style layout. For frequencies above 240 MHz, you can choose either a leveled or balanced-T or Y topology, as the leveled PHY calibrates to either implementation. Regardless of protocol, for devices without a levelling block—such as Arria II GZ, Arria V, and Cyclone V—a balanced-T PCB topology for address/command/clock must be used because fly-by topology is not supported.

For details about leveling delay chains, consult the memory interfaces hardware section of the device handbook for your FPGA.

The following figure shows the write datapath for a leveling interface. The full-rate PLL output clock `phy_write_clk` goes to a leveling delay chain block which generates all other periphery clocks that are needed. The data signals that generate DQ and DQS signals pass to an output phase alignment block. The output phase alignment block feeds an output buffer which creates a pair of pseudo differential clocks that connect to the memory. In full-rate designs, only the SDR-DDR portion of the path is used; in half-rate mode, the HDR-SDR circuitry is also required. The use of `DDIO_OUT` in both the output strobe and output data generation paths ensures that their timing characteristics are as similar as possible. The

`<variation_name>.pin_assignments.tcl` script automatically specifies the logic option that associates all data pins to the output strobe pin. The Quartus Prime Fitter treats the pins as a DQS/DQ pin group.

**Figure 4. Write Datapath for a Leveling Interface**



## 1.6. Read Datapath

The read datapath passes read data from memory to the PHY. The following figure shows the blocks and flow in the read datapath.

For all protocols, the DQS logic block delays the strobe by 90 degrees to center-align the rising strobe edge within the data window. For DDR2, DDR3, and LPDDR2 protocols, the logic block also performs strobe gating, holding the DQS enable signal high for the entire period that data is received. One DQS logic block exists for each data group.

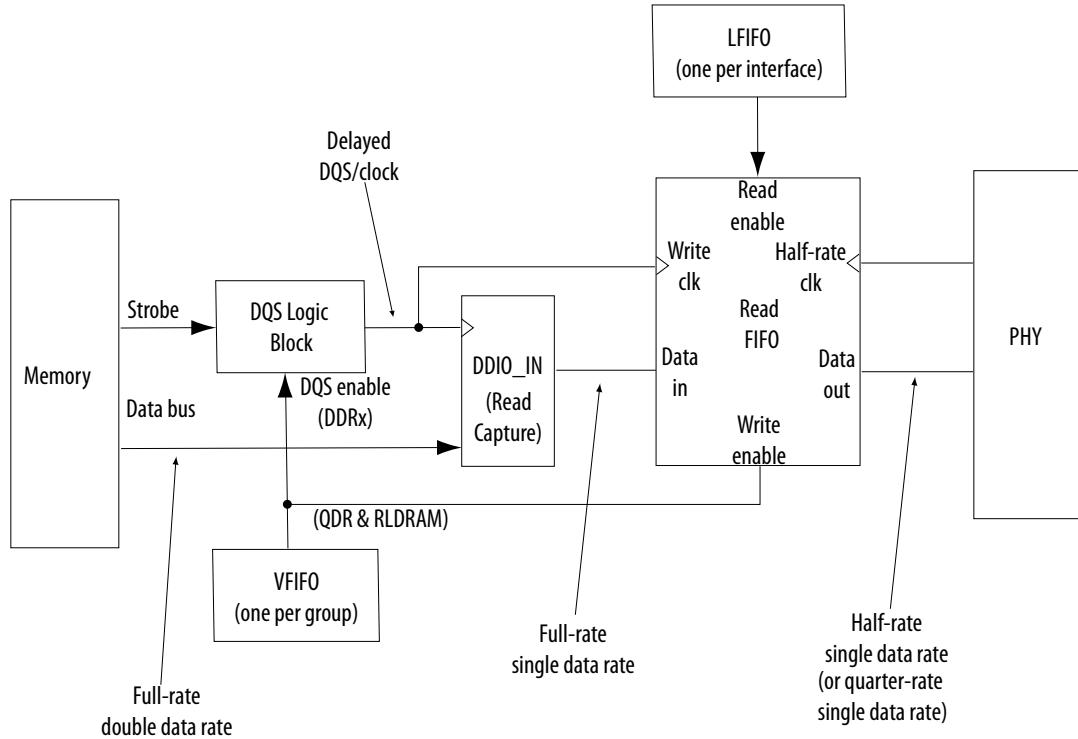
One VFIFO buffer exists for each data group. For DDR2, DDR3, and LPDDR2 protocols, the VFIFO buffer generates the DQS enable signal, which is delayed (by an amount determined during calibration) to align with the incoming DQS signal. For QDR and RLDRAM protocols, the output of the VFIFO buffer serves as the write enable signal for the Read FIFO buffer, signaling when to begin capturing data.

DDIO\_IN receives data from memory at double-data rate and passes data on to the Read FIFO buffer at single-data rate.

The Read FIFO buffer temporarily holds data read from memory; one Read FIFO buffer exists for each data group. For half-rate interfaces, the Read FIFO buffer converts the full-rate, single data-rate input to a half-rate, single data-rate output which is then passed to the PHY core logic. In the case of a quarter-rate interface, soft logic in the PHY performs an additional conversion from half-rate single data rate to quarter-rate single data rate.

One LFIFO buffer exists for each memory interface; the LFIFO buffer generates the read enable signal for all Read FIFO blocks in an interface. The read enable signal is asserted when the Read FIFO blocks have buffered sufficient data from the memory to be read. The timing of the read enable signal is determined during calibration.

**Figure 5. Read Datapath**



## 1.7. Sequencer

Depending on the combination of protocol and IP architecture in your external memory interface, you may have either an RTL-based sequencer or a Nios® II-based sequencer.



RTL-based sequencer implementations and Nios II-based sequencer implementations can have different pin requirements. You may not be able to migrate from an RTL-based sequencer to a Nios II-based sequencer and maintain the same pinout.

For information on sequencer support for different protocol-architecture combinations, refer to *Introduction to Intel® FPGA Memory Solutions* in Volume 1 of this handbook. For information on pin planning, refer to *Planning Pin and FPGA Resources* in Volume 2 of this handbook.

#### Related Information

- [Protocol Support Matrix](#)
- [Planning Pin and FPGA Resources](#)

### 1.7.1. Nios II-Based Sequencer

The DDR2, DDR3, and LPDDR2 controllers with UniPHY employ a Nios II-based sequencer that is parameterizable and is dynamically generated at run time. The Nios II-based sequencer is also available with the QDR II and RLDRAM II controllers.

#### 1.7.1.1. NIOS II-based Sequencer Function

The sequencer enables high-frequency memory interface operation by calibrating the interface to compensate for variations in setup and hold requirements caused by transmission delays.

UniPHY converts the double-data rate interface of high-speed memory devices to a full-rate or half-rate interface for use within an FPGA. To compensate for slight variations in data transmission to and from the memory device, double-data rate is usually center-aligned with its strobe signal; nonetheless, at high speeds, slight variations in delay can result in setup or hold time violations. The sequencer implements a calibration algorithm to determine the combination of delay and phase settings necessary to maintain center-alignment of data and clock signals, even in the presence of significant delay variations. Programmable delay chains in the FPGA I/Os then implement the calculated delays to ensure that data remains centered. Calibration also applies settings to the FIFO buffers within the PHY to minimize latency and ensures that the read valid signal is generated at the appropriate time.

When calibration is completed, the sequencer returns control to the memory controller.

For more information about calibration, refer to *UniPHY Calibration Stages*, in this chapter.

#### Related Information

- [UniPHY Calibration Stages](#) on page 55

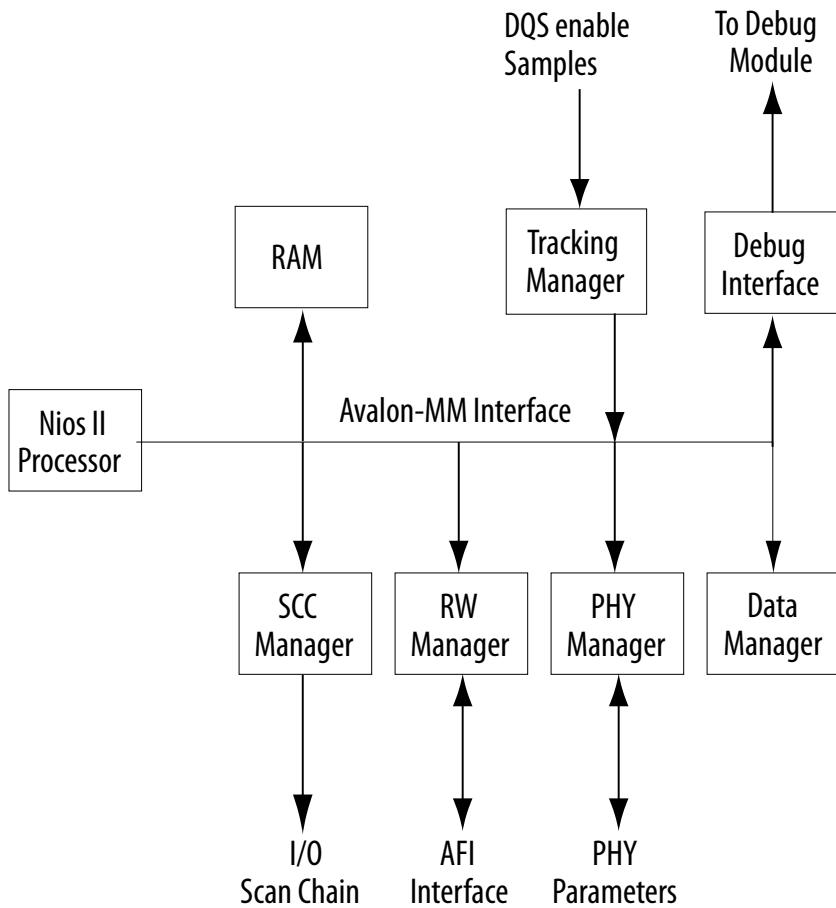
#### 1.7.1.2. Nios II-based Sequencer Architecture

The sequencer is composed of a Nios II processor and a series of hardware-based component managers, connected together by an Avalon bus. The Nios II processor performs the high-level algorithmic operations of calibration, while the component managers handle the lower-level timing, memory protocol, and bit-manipulation operations.



The high-level calibration algorithms are specified in C code, which is compiled into Nios II code that resides in the FPGA RAM blocks. The debug interface provides a mechanism for interacting with the various managers and for tracking the progress of the calibration algorithm, and can be useful for debugging problems that arise within the PHY. The various managers are specified in RTL and implement operations that would be slow or inefficient if implemented in software.

**Figure 6. NIOS II-based Sequencer Block Diagram**



The C code that defines the calibration routines is available for your reference in the \<name>\_s0\_software subdirectory. Intel recommends that you do not modify this C code.

#### 1.7.1.3. Nios II-based Sequencer SCC Manager

The scan chain control (SCC) manager allows the sequencer to set various delays and phases on the I/Os that make up the memory interface. The latest Intel device families provide dynamic delay chains on input, output, and output enable paths which can be reconfigured at runtime. The SCC manager provides the calibration routines access to these chains to add delay on incoming and outgoing signals. A master on the Avalon-MM interface may require the maximum allowed delay setting on input and output paths, and may set a particular delay value in this range to apply to the paths.



The SCC manager implements the Avalon-MM interface and the storage mechanism for all input, output, and phase settings. It contains circuitry that configures a DQ- or DQS-configuration block. The Nios II processor may set delay, phases, or register settings; the sequencer scans the settings serially to the appropriate DQ or DQS configuration block.

#### 1.7.1.4. NIOS II-based Sequencer RW Manager

The read write (RW) manager encapsulates the protocol to read and write to the memory device through the Altera PHY Interface (AFI). It provides a buffer that stores the data to be sent to and read from memory, and provides the following commands:

- Write configuration—configures the memory for use. Sets up burst lengths, read and write latencies, and other device specific parameters.
- Refresh—initiates a refresh operation at the DRAM. The command does not exist on SRAM devices. The sequencer also provides a register that determines whether the RW manager automatically generates refresh signals.
- Enable or disable multi-purpose register (MPR)—for memory devices with a special register that contains calibration specific patterns that you can read, this command enables or disables access to the register.
- Activate row—for memory devices that have both rows and columns, this command activates a specific row. Subsequent reads and writes operate on this specific row.
- Precharge—closes a row before you can access a new row.
- Write or read burst—writes or reads a burst length of data.
- Write guaranteed—writes with a special mode where the memory holds address and data lines constant. Intel guarantees this type of write to work in the presence of skew, but constrains to write the same data across the entire burst length.
- Write and read back-to-back—performs back-to-back writes or reads to adjacent banks. Most memory devices have strict timing constraints on subsequent accesses to the same bank, thus back-to-back writes and reads have to reference different banks.
- Protocol-specific initialization—a protocol-specific command required by the initialization sequence.

#### 1.7.1.5. NIOS II-based Sequencer PHY Manager

The PHY Manager provides access to the PHY for calibration, and passes relevant calibration results to the PHY. For example, the PHY Manager sets the VFIFO and LFIFO buffer parameters resulting from calibration, signals the PHY when the memory initialization sequence finishes, and reports the pass/fail status of calibration.

#### 1.7.1.6. NIOS II-based Sequencer Data Manager

The Data Manager stores parameterization-specific data in RAM, for the software to query.

#### 1.7.1.7. NIOS II-based Sequencer Tracking Manager

The Tracking Manager detects the effects of voltage and temperature variations that can occur on the memory device over time resulting in reduced margins, and adjusts the DQS enable delay as necessary to maintain adequate operating margins.

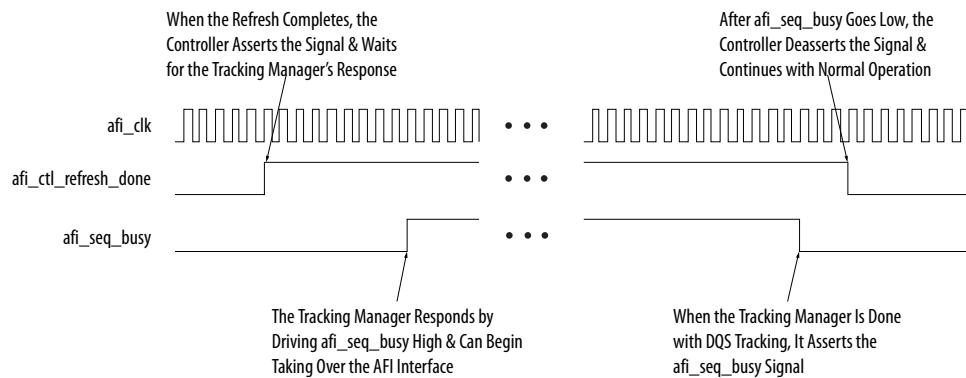


The Tracking Manager briefly assumes control of the AFI interface after each memory refresh cycle, issuing a read routine to the RW Manager, and then sampling the DQS tracking. Ideally, the falling edge of the DQS enable signal would align to the last rising edge of the raw DQS signal from the memory device. The Tracking Manager determines whether the DQS enable signal is leading or trailing the raw DQS signal.

Each time a refresh occurs, the Tracking Manager takes a sample of the raw DQS signal; any adjustments of the DQS enable signal occur only after sufficient samples of raw DQS have been taken. When the Tracking Manager determines that the DQS enable signal is either leading or lagging the raw DQS signal, it adjusts the DQS enable appropriately.

The following figure shows the Tracking manager signals.

**Figure 7. Tracking Manager Signals**



Some notes on Tracking Manager operation:

- The time taken by the Tracking Manager is arbitrary; if the period taken exceeds the refresh period, the Tracking Manager handles memory refresh.
- **afi\_seq\_busy** should go high fewer than 10 clock cycles after **afi\_ctl\_refresh\_done** or **afi\_ctl\_long\_idle** is asserted.
- **afi\_refresh\_done** should deassert fewer than 10 clock cycles after **afi\_seq\_busy** deasserts.



- afi\_ctl\_long\_idle causes the Tracking Manager to execute an algorithm different than periodic refresh; use afi\_ctl\_long\_idle when a long session has elapsed without a periodic refresh.
- The Tracking Manager is instantiated into the sequencer system when DQS Tracking is turned on.

**Table 1. Configurations Supporting DQS Tracking**

| Device Family                      | Protocol             | Memory Clock Frequency                                       |
|------------------------------------|----------------------|--|
| Arria V (GX/GT/SX/ST) , Cyclone V  | LPDDR2 (single rank) | All frequencies.   |
| Arria V (GX/GT/SX/ST)              | DDR3 (single rank)   | 450 MHz or higher for speed grade 5, or higher than 534 MHz. |
| Arria V GZ, Stratix V (E/GS/GT/GX) |                      | 750 MHz or higher.   |

- If you do not want to use DQS tracking, you can disable it (at your own risk), by opening the Verilog file <variant\_name>\_if0\_c0.v in an editor, and changing the value of the USE\_DQS\_TRACKING parameter from 1 to 0.

### 1.7.1.8. NIOS II-based Sequencer Processor

The Nios II processor manages the calibration algorithm; the Nios II processor is unavailable after calibration is completed.

The same calibration algorithm supports all device families, with some differences. The following sections describe the calibration algorithm for DDR3 SDRAM on Stratix III devices. Calibration algorithms for other protocols and families are a subset and significant differences are pointed out when necessary. As the algorithm is fully contained in the software of the sequencer (in the C code) enabling and disabling specific steps involves turning flags on and off.

Calibration consists of the following stages:

- Initialize memory.
- Calibrate read datapath.
- Calibrate write datapath.
- Run diagnostics.

### 1.7.1.9. NIOS II-based Sequencer Calibration and Diagnostics

Calibration must initialize all memory devices before they can operate properly. The sequencer performs this memory initialization stage when it takes control of the PHY at startup.

Calibrating the read datapath comprises the following steps:

- Calibrate DQS enable cycle and phase.
- Perform read per-bit deskew to center the strobe signal within data valid window.
- Reduce LFIFO latency.

Calibrating the write datapath involves the following steps:

- Center align DQS with respect to DQ.
- Align DQS with mem\_clk.



The sequencer estimates the read and write margins under noisy conditions, by sweeping input and output DQ and DQS delays to determine the size of the data valid windows on the input and output sides. The sequencer stores this diagnostic information in the local memory and you can access it through the debugging interface.

When the diagnostic test finishes, control of the PHY interface passes back to the controller and the sequencer issues a pass or fail signal.

#### Related Information

[External Memory Interface Debug Toolkit](#) on page 471

### 1.7.2. RTL-based Sequencer

The RTL-based sequencer is available for QDR II and RLDRAM II interfaces, on supported device families other than Arria V. The RTL sequencer is a state machine that processes the calibration algorithm.

The sequencer assumes control of the interface at reset (whether at initial startup or when the IP is reset) and maintains control throughout the calibration process. The sequencer relinquishes control to the memory controller only after successful calibration. The following tables list the major states in the RTL-based sequencer.

**Table 2. Sequencer States**

| RTL-based Sequencer State    | Description  |
|------------------------------|--|
| RESET                        | Remain in this state until reset is released.  |
| LOAD_INIT                    | Load any initialization values for simulation purposes.  |
| STABLE                       | Wait until the memory device is stable.  |
| WRITE_ZERO                   | Issue write command to address 0.  |
| WAIT_WRITE_ZERO              | Write all 0xAs to address 0.   |
| WRITE_ONE                    | Issue write command to address 1.  |
| WAIT_WRITE_ONE               | Write all 0x5s to address 1.   |
| Valid Calibration States     |  |
| V_READ_ZERO                  | Issue read command to address 0 (expected data is all 0xAs).   |
| V_READ_NOP                   | This state represents the minimum number of cycles required between 2 back-to-back read commands. The number of NOP states depends on the burst length.  |
| V_READ_ONE                   | Issue read command to address 1 (expected data is all 0x5s).   |
| V_WAIT_READ                  | Wait for read valid signal.  |
| V_COMPARE_READ_ZERO_READ_ONE | Parameterizable number of cycles to wait before making the read data comparisons.  |
| V_CHECK_READ_FAIL            | When a read fails, the write pointer (in the AFI clock domain) of the valid FIFO buffer is incremented. The read pointer of the valid FIFO buffer is in the DQS clock domain. The gap between the read and write pointers is effectively the latency between the time when the PHY receives the read command and the time valid data is returned to the PHY. |
| V_ADD_FULL_RATE              | Advance the read valid FIFO buffer write pointer by an extra full rate cycle.  |
| V_ADD_HALF_RATE              | Advance the read valid FIFO buffer write pointer by an extra half rate cycle. In full-rate designs, equivalent to V_ADD_FULL_RATE.   |

*continued...*

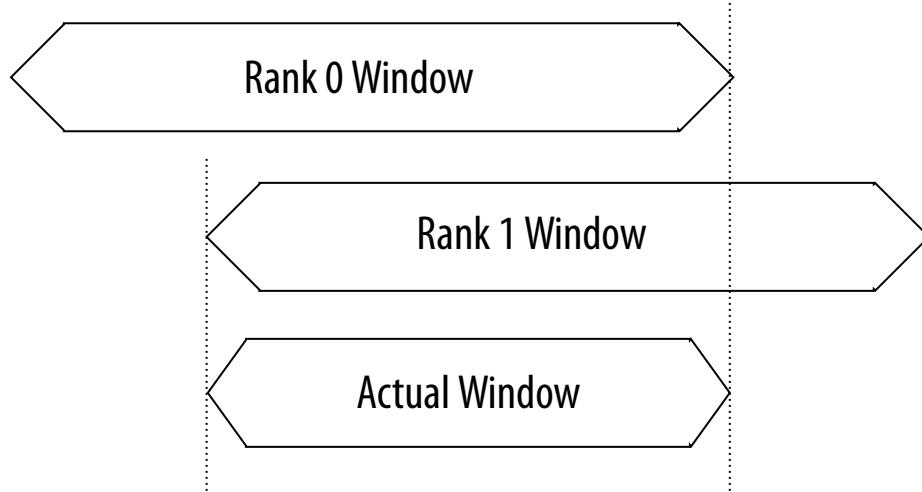
| <b>RTL-based Sequencer State</b>  | <b>Description</b>   |
|-----------------------------------|--|
| V_READ_FIFO_RESET                 | Reset the read and write pointers of the read data synchronization FIFO buffer.  |
| V_CALIB_DONE                      | Valid calibration is successful.   |
| <b>Latency Calibration States</b> |  |
| L_READ_ONE                        | Issue read command to address 1 (expected data is all 0x5s).   |
| L_WAIT_READ                       | Wait for read valid signal from read datapath. Initial read latency is set to a predefined maximum value.  |
| L_COMPARE_READ_ONE                | Check returned read data against expected data. If data is correct, go to L_REDUCE_LATENCY; otherwise go to L_ADD_MARGIN.  |
| L_REDUCE_LATENCY                  | Reduce the latency counter by 1.   |
| L_READ_FLUSH                      | Read from address 0, to flush the contents of the read data resynchronization FIFO buffer.   |
| L_WAIT_READ_FLUSH                 | Wait until the whole FIFO buffer is flushed, then go back to L_READ and try again.   |
| L_ADD_MARGIN                      | Increment latency counter by 3 (1 cycle to get the correct data, 2 more cycles of margin for run time variations). If latency counter value is smaller than predefined ideal condition minimum, then go to CALIB_FAIL. |
| CALIB_DONE                        | Calibration is successful.   |
| CALIB_FAIL                        | Calibration is not successful.   |

## 1.8. Shadow Registers

Shadow registers are a hardware feature of Arria V GZ and Stratix V devices that enables high-speed multi-rank calibration for DDR3 quarter-rate and half-rate memory interfaces, up to 800MHz for dual-rank interfaces and 667MHz for quad-rank interfaces.

Prior to the introduction of shadow registers, the data valid window of a multi-rank interface was calibrated to the overlapping portion of the data valid windows of the individual ranks. The resulting data valid window for the interface would be smaller than the individual data valid windows, limiting overall performance.

**Figure 8. Calibration of Overlapping Data Valid Windows, without Shadow Registers**

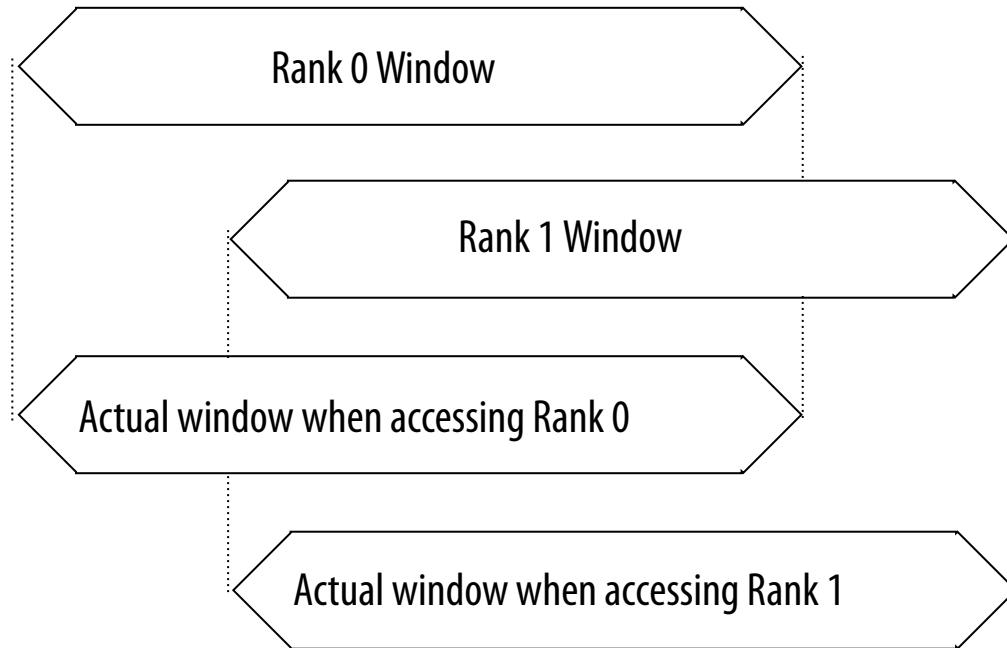




Shadow registers allow the sequencer to calibrate each rank separately and fully, and then to save the calibrated settings for each rank in its own set of shadow registers, which are part of the IP scan chains. During a rank-to-rank switch, the rank-specific set of calibration settings is restored just-in-time to optimize the data valid window for each rank.

The following figure illustrates how the use of rank-specific calibration settings results in a data valid window appropriate for the current rank.

**Figure 9. Rank-specific Calibration Settings, with Shadow Registers**



The shadow registers and their associated rank-switching circuitry are part of the device I/O periphery hardware.

### 1.8.1. Shadow Registers Operation

The sequencer calibrates each rank individually and stores the resulting configuration in shadow registers, which are part of the IP scan chains. UniPHY then selects the appropriate configuration for the rank in use, switching between configurations as necessary. Calibration results for deskew delay chains are stored in the shadow registers. For DQS enable/disable, delay chain configurations come directly from the FPGA core.

#### Signals

The afi\_wrrank signal indicates the rank to which the controller is writing, so that the PHY can switch to the appropriate setting. Signal timing is identical to afi\_dqs\_burst; that is, afi\_wrrank must be asserted at the same time as afi\_dqs\_burst, and must be of the same duration.

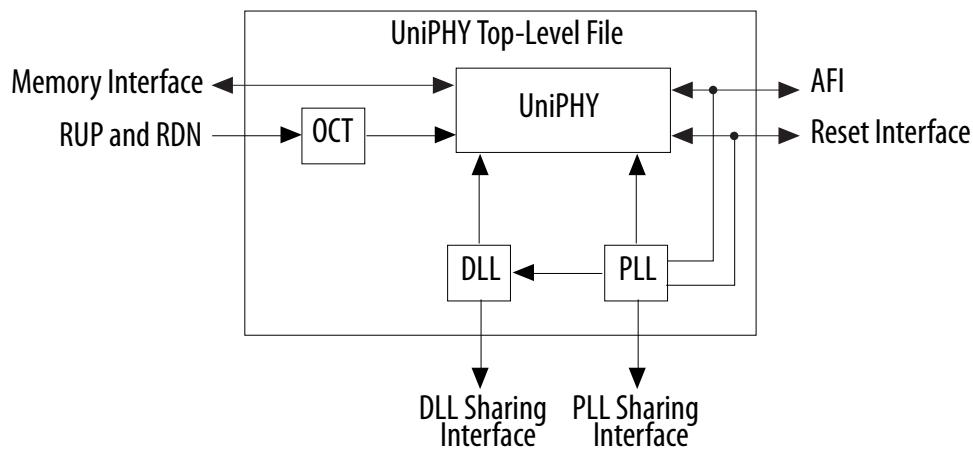
The afi\_rrank signal indicates the rank from which the controller is reading, so that the PHY can switch to the appropriate setting. This signal must be asserted at the same time as afi\_rdata\_en when issuing a read command, and once asserted, must remain unchanged until the controller issues a new read command to another rank.

## 1.9. UniPHY Interfaces

The following figure shows the major blocks of the UniPHY and how it interfaces with the external memory device and the controller.

**Note:** Instantiating the delay-locked loop (DLL) and the phase-locked loop (PLL) on the same level as the UniPHY eases DLL and PLL sharing.

**Figure 10. UniPHY Interfaces with the Controller and the External Memory**



The following interfaces are on the UniPHY top-level file:

- AFI
- Memory interface
- DLL sharing interface
- PLL sharing interface
- OCT interface

### AFI

The UniPHY datapath uses the Altera PHY interface (AFI). The AFI is in a simple connection between the PHY and controller. The AFI is based on the DDR PHY interface (DFI) specification, with some calibration-related signals not used and some additional Intel-specific sideband signals added.

For more information about the AFI, refer to *AFI 4.0 Specification*, in this chapter.

### The Memory Interface

For information on the memory interface, refer to *UniPHY Signals*, in this chapter.

### Related Information

- [AFI 4.0 Specification](#) on page 117



- [UniPHY Signals](#) on page 32

### 1.9.1. The DLL and PLL Sharing Interface

You can generate the UniPHY memory interface and configure it to share its PLL, DLL, or both interfaces.

By default, a UniPHY memory interface variant contains a PLL and DLL; the PLL produces a variety of required clock signals derived from the reference clock, and the DLL produces a delay codeword. In this case the PLL sharing mode is "No sharing". A UniPHY variant can be configured as a PLL Master and/or DLL Master, in which case the corresponding interfaces are exported to the UniPHY top-level and can be connected to an identically configured UniPHY variant PLL Slave and/or DLL Slave. The UniPHY slave variant is instantiated without a PLL and/or DLL, which saves device resources.

**Note:** For Arria II GX, Arria II GZ, Stratix III, and Stratix IV devices, the PLL and DLL must both be shared at the same time—their sharing modes must match. This restriction does not apply to Arria V, Arria V GZ, Cyclone V, or Stratix V devices.

**Note:** For devices with hard memory interface components onboard, you cannot share PLL or DLL resources between soft and hard interfaces.

#### 1.9.1.1. Sharing PLLs or DLLs

To share PLLs or DLLs, follow these steps:

1. To create a PLL or DLL master, create a UniPHY memory interface IP core. To make the PLL and/or DLL interface appear at the top-level in the core, on the **PHY Settings** tab in the parameter editor, set the **PLL Sharing Mode** and/or **DLL Sharing Mode** to Master.
2. To create a PLL or DLL slave, create a second UniPHY memory interface IP core. To make the PLL and/or DLL interface appear at the top-level in the core, on the **PHY Settings** tab set the **PLL Sharing Mode** and/or **DLL Sharing Mode** to Slave.
3. Connect the PLL and/or DLL sharing interfaces by following the appropriate step, below:
  - **For cores generated with IP Catalog** : connect the PLL and/or DLL interface ports between the master and slave cores in your wrapper RTL. When using PLL sharing, connect the afi\_clk, afi\_half\_clk, and afi\_reset\_export\_n outputs from the UniPHY PLL master to the afi\_clk, afi\_half\_clk, and afi\_reset\_in inputs on the UniPHY PLL slave.
  - **For cores generated with Qsys** , connect the PLL and/or DLL interface in the Qsys GUI. When using PLL sharing, connect the afi\_clk, afi\_half\_clk, and afi\_reset\_export\_n outputs from the UniPHY PLL master to the afi\_clk, afi\_half\_clk, and afi\_reset\_in inputs on the UniPHY PLL slave.

Qsys supports only one-to-one conduit connections in the patch panel. To share a PLL from a UniPHY PLL master with multiple slaves, you should replicate the number of PLL sharing conduit interfaces in the Qsys patch panel by choosing **Number of PLL sharing interfaces** in the parameter editor.



**Note:** You may connect a slave UniPHY instance to the clocks from a user-defined PLL instead of from a UniPHY master. The general procedure for doing so is as follows:

1. Make a template, by generating your IP with **PLL Sharing Mode** set to **No Sharing**, and then compiling the example project to determine the frequency and phases of the clock outputs from the PLL.
2. Generate an external PLL using the IP Catalog flow, with the equivalent output clocks.
3. Generate your IP with **PLL Sharing Mode** set to **Slave**, and connect the external PLL to the PLL sharing interface.

You must be very careful when connecting clock signals to the slave. Connecting to clocks with frequency or phase different than what the core expects may result in hardware failure.

**Note:** The signal `dll_pll_locked` is an internal signal from the PLL to the DLL which ensures that the DLL remains in reset mode until the PLL becomes locked. This signal is not available for use by customer logic.

### 1.9.1.2. About PLL Simulation

PLL frequencies may differ between the synthesis and simulation file sets. In either case the achieved PLL frequencies and phases are calculated and reported in real time in the parameter editor.

For the simulation file set, clocks are specified in the RTL, not in units of frequency but by the period in picoseconds, thus avoiding clock drift due to picosecond rounding error.

For the synthesis file set, there are two mechanisms by which clock frequencies are specified in the RTL, based on the target device family:

- For Arria V, Arria V GZ, Cyclone V, and Stratix V, clock frequencies are specified in MHz.
- For Arria II GX, Arria II GZ, Stratix III, and Stratix IV, clock frequencies are specified by integer multipliers and divisors. For these families, the real simulation model—as opposed to the default abstract simulation model—also uses clock frequencies specified by integer ratios.

### 1.9.2. The OCT Sharing Interface

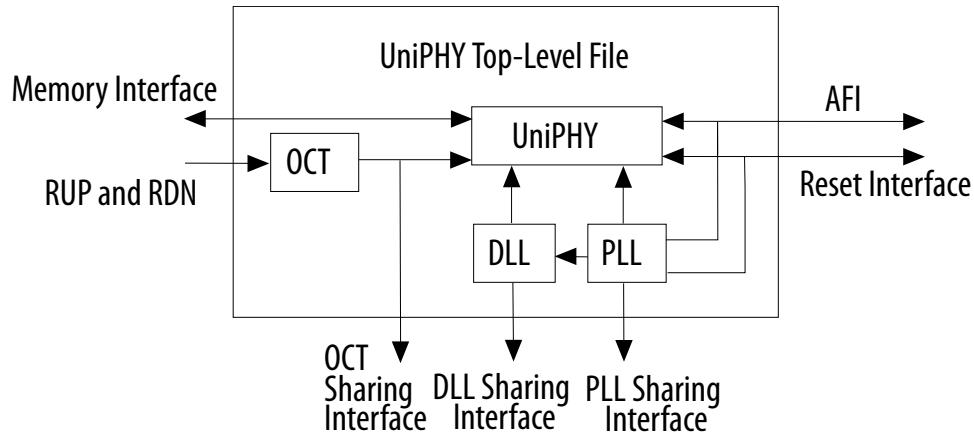
By default, the UniPHY IP generates the required OCT control block at the top-level RTL file for the PHY.

If you want, you can instantiate this block elsewhere in your code and feed the required termination control signals into the IP core by turning off **Master for OCT Control Block** on the **PHY Settings** tab. If you turn off **Master for OCT Control Block**, you must instantiate the OCT control block or use another UniPHY instance as a master, and ensure that the parallel and series termination control bus signals are connected to the PHY.

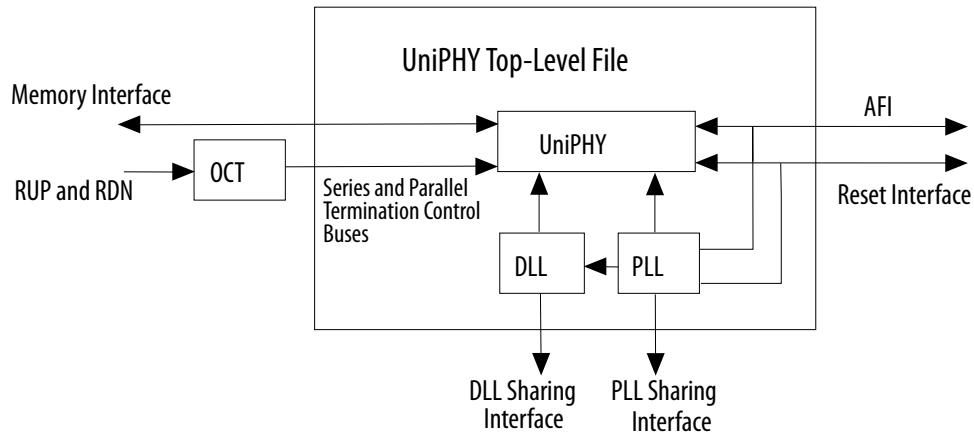
The following figures show the PHY architecture with and without Master for OCT Control Block.



**Figure 11. PHY Architecture with Master for OCT Control Block**



**Figure 12. PHY Architecture without Master for OCT Control Block**



### 1.9.2.1. Modifying the Pin Assignment Script for QDR II and RLDRAM II

If you generate a QDR II or RLDRAM II slave IP core, you must modify the pin assignment script to allow the fitter to correctly resolve the OCT termination block name in the OCT master core.

To modify the pin assignment script for QDR II or RLDRAM II slaves, follow these steps:

1. In a text editor, open your system's Tcl pin assignments script file, as follows:
  - For systems generated with the IP Catalog: Open the `<IP core name>/<slave core name>_p0_pin_assignments.tcl` file.
  - For systems generated with Qsys: Open the `<HDL Path>/<submodules>/<slave core name>_p0_pin_assignments.tcl` file.
2. Search for the following line:

```
set ::master_corename "_MASTER_CORE_"
```



3. Replace \_MASTER\_CORE\_ with the instance name of the UniPHY master to which the slave is connected. The instance name is determined from the pin assignments file name, as follows:
  - For systems generated with Qsys, the instance name is the <master core name> component of the pins assignments file name: <HDL path>/<submodules>/<master core name>\_p0\_pin\_assignments.tcl.
  - For systems generated with the IP Catalog, the instance name is the <master core name> component of the pins assignments file name: <IP core name>/<master core name>\_p0\_pin\_assignments.tcl .

## 1.10. UniPHY Signals

The following tables list the UniPHY signals.

**Table 3. Clock and Reset Signals**

| Name           | Direction | Width | Description  |
|----------------|-----------|-------|--|
| pll_ref_clk    | Input     | 1     | PLL reference clock input.   |
| global_reset_n | Input     | 1     | Active low global reset for PLL and all logic in the PHY, which causes a complete reset of the whole system. Minimum recommended pulse width is 100ns.     |
| soft_reset_n   | Input     | 1     | Holding soft_reset_n low holds the PHY in a reset state. However it does not reset the PLL, which keeps running. It also holds the afi_reset_n output low. |

**Table 4. DDR2 and DDR3 SDRAM Interface Signals**

| Name               | Direction     | Width                  | Description            |
|--------------------|---------------|------------------------|------------------------|
| mem_ck, mem_ck_n   | Output        | MEM_CK_WIDTH           | Memory clock.          |
| mem_cke            | Output        | MEM_CLK_EN_WIDTH       | Clock enable.          |
| mem_cs_n           | Output        | MEM_CHIP_SELECT_WIDTH  | Chip select..          |
| mem_cas_n          | Output        | MEM_CONTROL_WIDTH      | Column address strobe. |
| mem_ras_n          | Output        | MEM_CONTROL_WIDTH      | Row address strobe.    |
| mem_we_n           | Output        | MEM_CONTROL_WIDTH      | Write enable.          |
| mem_a              | Output        | MEM_ADDRESS_WIDTH      | Address.               |
| mem_ba             | Output        | MEM_BANK_ADDRESS_WIDTH | Bank address.          |
| mem_dqs, mem_dqs_n | Bidirectional | MEM_DQS_WIDTH          | Data strobe.           |
| mem_dq             | Bidirectional | MEM_DQ_WIDTH           | Data.                  |
| mem_dm             | Output        | MEM_DM_WIDTH           | Data mask.             |

*continued...*



| Name   | Direction | Width             | Description  |
|--|-----------|-------------------|--|
| mem_odt                                      | Output    | MEM_ODT_WIDTH     | On-die termination.  |
| mem_reset_n (DDR3 only)                      | Output    | 1                 | Reset  |
| mem_ac_parity (DDR3 only, RDIMM/LRDIMM only) | Output    | MEM_CONTROL_WIDTH | Address/command parity bit. (Even parity, per the RDIMM spec, JESD82-29A.) |
| mem_err_out_n (DDR3 only, RDIMM/LRDIMM only) | Input     | MEM_CONTROL_WIDTH | Address/command parity error.  |

**Table 5. UniPHY Parameters**

| Parameter Name                  | Description   |
|---------------------------------|---|
| AFI_RATIO                       | AFI_RATIO is 1 in full-rate designs.<br>AFI_RATIO is 2 for half-rate designs.<br>AFI_RATIO is 4 for quarter-rate designs. |
| MEM_IF_DQS_WIDTH                | The number of DQS pins in the interface.  |
| MEM_ADDRESS_WIDTH               | The address width of the specified memory device.   |
| MEM_BANK_WIDTH                  | The bank width of the specified memory device.  |
| MEM_CHIP_SELECT_WIDTH           | The chip select width of the specified memory device.   |
| MEM_CONTROL_WIDTH               | The control width of the specified memory device.   |
| MEM_DM_WIDTH                    | The DM width of the specified memory device.  |
| MEM_DQ_WIDTH                    | The DQ width of the specified memory device.  |
| MEM_READ_DQS_WIDTH              | The READ DQS width of the specified memory device.  |
| MEM_WRITE_DQS_WIDTH             | The WRITE DQS width of the specified memory device.   |
| OCT_SERIES_TERM_CONTROL_WIDTH   | —   |
| OCT_PARALLEL_TERM_CONTROL_WIDTH | —   |
| AFI_ADDRESS_WIDTH               | The AFI address width, derived from the corresponding memory interface width.   |
| AFI_BANK_WIDTH                  | The AFI bank width, derived from the corresponding memory interface width.  |
| AFI_CHIP_SELECT_WIDTH           | The AFI chip select width, derived from the corresponding memory interface width.   |
| AFI_DATA_MASK_WIDTH             | The AFI data mask width.  |
| AFI_CONTROL_WIDTH               | The AFI control width, derived from the corresponding memory interface width.   |
| AFI_DATA_WIDTH                  | The AFI data width.   |

*continued...*



| Parameter Name                   | Description  |
|----------------------------------|--|
| AFI_DQS_WIDTH                    | The AFI DQS width.   |
| DLL_DELAY_CTRL_WIDTH             | The DLL delay output control width.  |
| NUM_SUBGROUP_PER_READ_DQS        | A read datapath parameter for timing purposes.                                 |
| QVLD_EXTRA_FLOP_STAGES           | A read datapath parameter for timing purposes.                                 |
| READ_VALID_TIMEOUT_WIDTH         | A read datapath parameter; calibration fails when the timeout counter expires. |
| READ_VALID_FIFO_WRITE_ADDR_WIDTH | A read datapath parameter; the write address width for half-rate clocks.       |
| READ_VALID_FIFO_READ_ADDR_WIDTH  | A read datapath parameter; the read address width for full-rate clocks.        |
| MAX_LATENCY_COUNT_WIDTH          | A latency calibration parameter; the maximum latency count width.              |
| MAX_READ_LATENCY                 | A latency calibration parameter; the maximum read latency.                     |
| READ_FIFO_READ_ADDR_WIDTH        | —  |
| READ_FIFO_WRITE_ADDR_WIDTH       | —  |
| MAX_WRITE_LATENCY_COUNT_WIDTH    | A write datapath parameter; the maximum write latency count width.             |
| INIT_COUNT_WIDTH                 | An initialization sequence.  |
| MRSC_COUNT_WIDTH                 | A memory-specific initialization parameter.                                    |
| INIT_NOP_COUNT_WIDTH             | A memory-specific initialization parameter.                                    |
| MRS_CONFIGURATION                | A memory-specific initialization parameter.                                    |
| MRS_BURST_LENGTH                 | A memory-specific initialization parameter.                                    |
| MRS_ADDRESS_MODE                 | A memory-specific initialization parameter.                                    |
| MRS_DLL_RESET                    | A memory-specific initialization parameter.                                    |
| MRS_IMP_MATCHING                 | A memory-specific initialization parameter.                                    |
| MRS_ODT_EN                       | A memory-specific initialization parameter.                                    |
| MRS_BURST_LENGTH                 | A memory-specific initialization parameter.                                    |
| MEM_T_WL                         | A memory-specific initialization parameter.                                    |
| MEM_T_RL                         | A memory-specific initialization parameter.                                    |
| SEQ_BURST_COUNT_WIDTH            | The burst count width for the sequencer.                                       |
| VCALIB_COUNT_WIDTH               | The width of a counter that the sequencer uses.                                |

*continued...*



| Parameter Name      | Description                                   |
|---------------------|---|
| DOUBLE_MEM_DQ_WIDTH | —   |
| HALF_AFI_DATA_WIDTH | —   |
| CALIB_REG_WIDTH     | The width of the calibration status register. |
| NUM_AFI_RESET       | The number of AFI resets to generate.         |

**Note:** For information about the AFI signals, refer to *AFI 4.0 Specification* in this chapter.

#### Related Information

[AFI 4.0 Specification](#) on page 117

## 1.11. PHY-to-Controller Interfaces

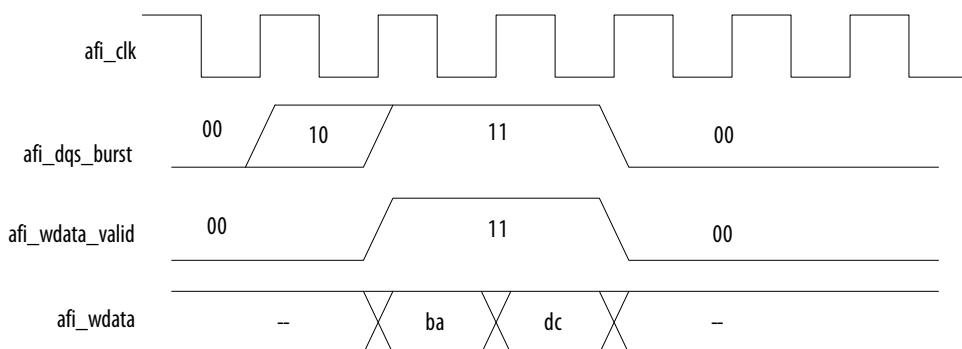
Various modules connect to UniPHY through specific ports.

The AFI standardizes and simplifies the interface between controller and PHY for all Intel memory designs, thus allowing you to easily interchange your own controller code with Intel's high-performance controllers. The AFI PHY interface includes an administration block that configures the memory for calibration and performs necessary accesses to mode registers that configure the memory as required.

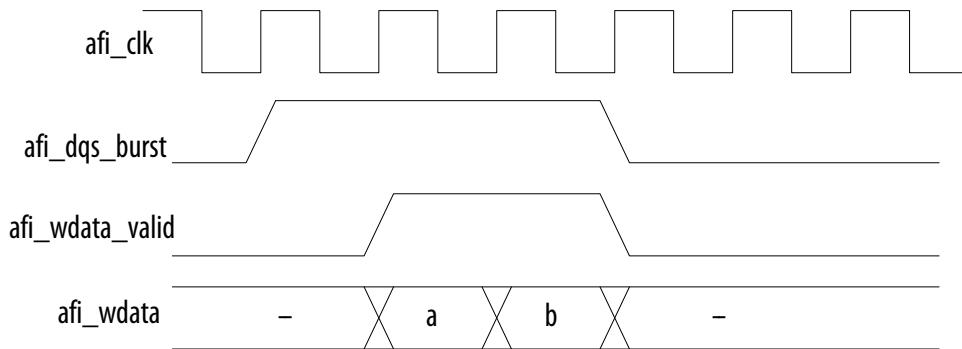
For half-rate designs, the address and command signals in the UniPHY are asserted for one mem\_clk cycle (1T addressing), such that there are two input bits per address and command pin in half-rate designs. If you require a more conservative 2T addressing (where signals are asserted for two mem\_clk cycles), drive both input bits (of the address and command signal) identically in half-rate designs.

The following figure shows the half-rate write operation.

**Figure 13. Half-Rate Write with Word-Aligned Data**



The following figure shows a full-rate write.

**Figure 14. Full-Rate Write**


For quarter-rate designs, the address and command signals in the UniPHY are asserted for one mem\_clk cycle (1T addressing), such that there are four input bits per address and command pin in quarter-rate designs. If you require a more conservative 2T addressing (where signals are asserted for two mem\_clk cycles), drive either the two lower input bits or the two upper input bits (of the address and command signal) identically.

After calibration is completed, the sequencer sends the write latency in number of clock cycles to the controller.

The AFI has the following conventions:

- With the AFI, high and low signals are combined in one signal, so for a single chip select (afi\_cs\_n) interface, afi\_cs\_n[1:0], location 0 appears on the memory bus on one mem\_clk cycle and location 1 on the next mem\_clk cycle.

*Note:* This convention is maintained for all signals so for an 8 bit memory interface, the write data (afi\_wdata) signal is afi\_wdata[31:0], where the first data on the DQ pins is afi\_wdata[7:0], then afi\_wdata[15:8], then afi\_wdata[23:16], then afi\_wdata[31:24].

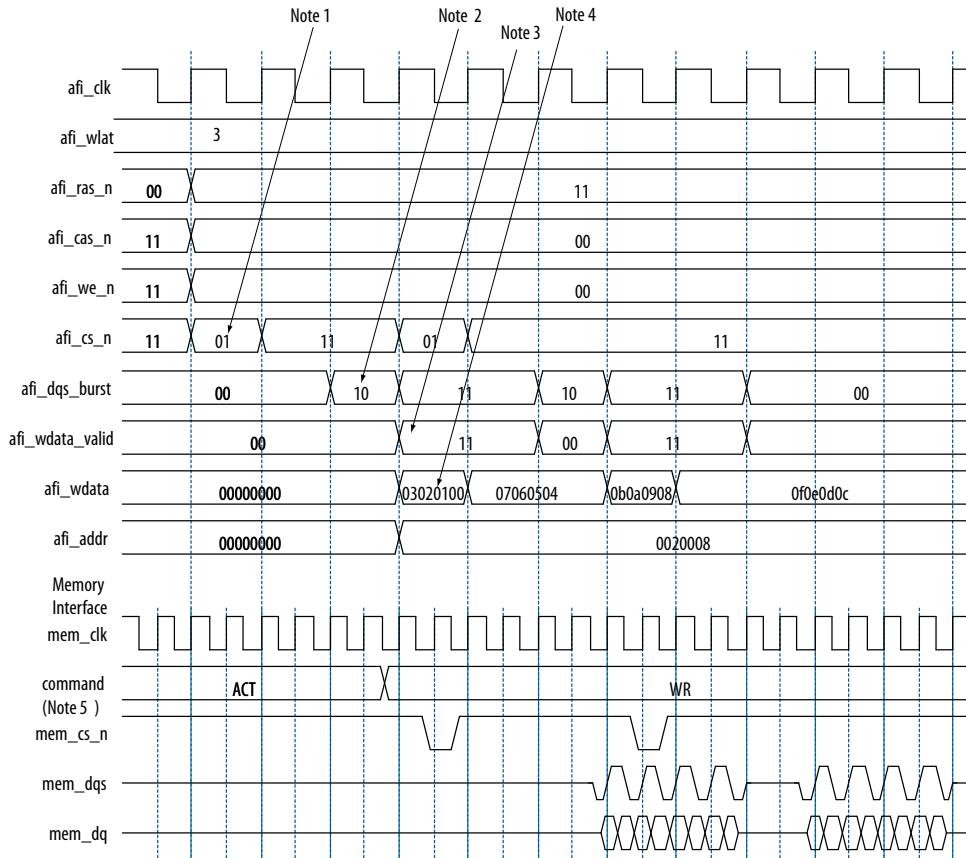
- Spaced reads and writes have the following definitions:
  - Spaced writes—write commands separated by a gap of one controller clock (afi\_clk) cycle.
  - Spaced reads—read commands separated by a gap of one controller clock (afi\_clk) cycle.

The following figures show writes and reads, where the IP core writes data to and reads from the same address. In each example, afi\_rdata and afi\_wdata are aligned with controller clock (afi\_clk) cycles. All the data in the bit vector is valid at once. These figures assume the following general points:

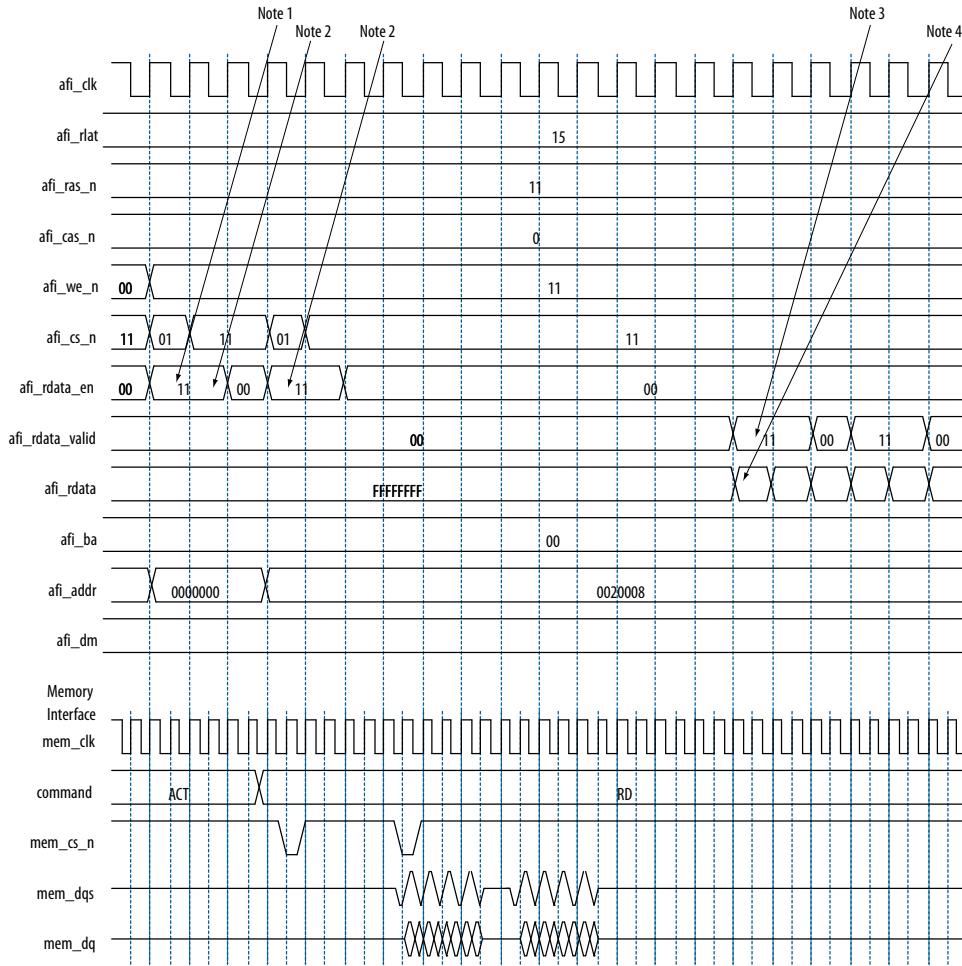
- The burst length is four.
- An 8-bit interface with one chip select.
- The data for one controller clock (afi\_clk) cycle represents data for two memory clock (mem\_clk) cycles (half-rate interface).



**Figure 15. Word-Aligned Writes**



## Figure 16. Word-Aligned Reads



## Notes to Figure:

1. For AFI, `afi_rdata_en` is required to be asserted one memory clock cycle before chip select (`afi_cs_n`) is asserted. In the half-rate `afi_clk` domain, this requirement manifests as the controller driving 11 (as opposed to the 01) on `afi_rdata_en`.
  2. AFI requires that `afi_rdata_en` is driven for the duration of the read. In this example, it is driven to 11 for two half-rate `afi_clks`, which equates to driving to 1, for the four memory clock cycles of this four-beat burst.
  3. The `afi_rdata_valid` returns 15 (`afi_rlat`) controller clock (`afi_clk`) cycles after `afi_rdata_en` is asserted. Returned is when the `afi_rdata_valid` signal is observed at the output of a register within the controller. A controller can use the `afi_rlat` value to determine when to register to returned data, but this is unnecessary as the `afi_rdata_valid` is provided for the controller to use as an enable when registering read data.
  4. Observe the alignment of returned read data with respect to data on the bus.

#### Related Information

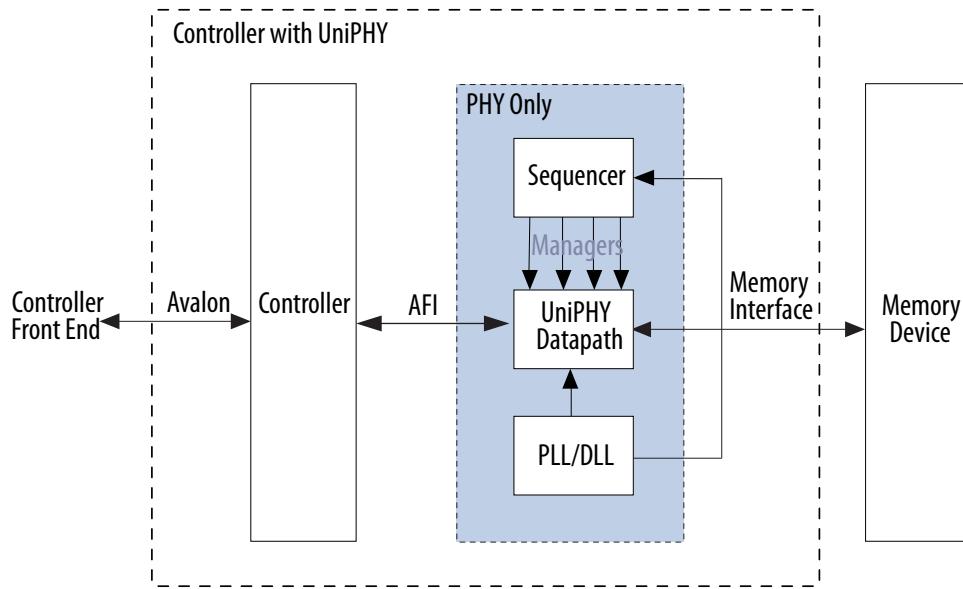
- [AFI 4.0 Specification](#) on page 117
- [Timing Diagrams for UniPHY IP](#) on page 435

## 1.12. Using a Custom Controller

By default, the UniPHY-based external memory interface IP cores are delivered with both the PHY and the memory controller integrated, as depicted in the following figure.

If you want to use your own custom controller with the UniPHY PHY, check the **Generate PHY only** box on the **PHY Settings** tab of the parameter editor and generate the IP. The resulting top-level IP consists of only the sequencer, UniPHY datapath, and PLL/DLL — the shaded area in the figure below.

**Figure 17. Memory Controller with UniPHY**



The AFI interface is exposed at the top-level of the generated IP core; you can connect the AFI interface to your custom controller.

When you enable **Generate PHY only**, the generated example designs include the memory controller appropriately instantiated to mediate read/write commands from the traffic generator to the PHY-only IP.

For information on the AFI protocol, refer to the *AFI 4.0 Specification*, in this chapter. For information on the example designs, refer to Chapter 9, *Example Designs*, in this volume.

#### Related Information

- [AFI 4.0 Specification](#) on page 117
- [Traffic Generator and BIST Engine](#) on page 409

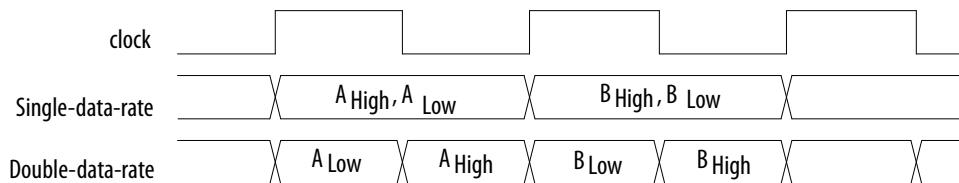
## 1.13. AFI 3.0 Specification

The Altera PHY interface (AFI) 3.0 defines communication between the controller and physical layer (PHY) in the external memory interface IP.

The AFI is a single-data-rate interface, meaning that data is transferred on the rising edge of each clock cycle. Most memory interfaces, however, operate at double-data-rate, transferring data on both the rising and falling edges of the clock signal. If the AFI interface is to directly control a double-data-rate signal, two single-data-rate bits must be transmitted on each clock cycle; the PHY then sends out one bit on the rising edge of the clock and one bit on the falling edge.

The AFI convention is to send the low part of the data first and the high part second, as shown in the following figure.

**Figure 18. Single Versus Double Data Rate Transfer**



### 1.13.1. Bus Width and AFI Ratio

In cases where the AFI clock frequency is one-half or one-quarter of the memory clock frequency, the AFI data must be twice or four times as wide, respectively, as the corresponding memory data. The ratio between AFI clock and memory clock frequencies is referred to as the AFI ratio. (A half-rate AFI interface has an AFI ratio of 2, while a quarter-rate interface has an AFI ratio of 4.)

In general, the width of the AFI signal depends on the following three factors:

- The size of the equivalent signal on the memory interface. For example, if `a[15:0]` is a DDR3 address input and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_addr` bus will be 16-bits wide.
- The data rate of the equivalent signal on the memory interface. For example, if `d[7:0]` is a double-data-rate QDR II input data bus and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_write_data` bus will be 16-bits wide.
- The AFI ratio. For example, if `cs_n` is a single-bit DDR3 chip select input and the AFI clock runs at half the speed of the memory interface, the equivalent `afi_cs_n` bus will be 2-bits wide.

The following formula summarizes the three factors described above:

```
AFI_width = memory_width * signal_rate * AFI_RATE_RATIO
```

**Note:** The above formula is a general rule, but not all signals obey it. For definite signal-size information, refer to the specific table.



### 1.13.2. AFI Parameters

The following tables list Altera PHY interface (AFI) parameters for AFI 4.0.

The parameters described in the following tables affect the width of AFI signal buses. Parameters prefixed by `MEM_IF_` refer to the signal size at the interface between the PHY and memory device.

**Table 6. Ratio Parameters**

| Parameter Name               | Description  |
|------------------------------|--|
| <code>AFI_RATE_RATIO</code>  | The ratio between the AFI clock frequency and the memory clock frequency. For full-rate interfaces this value is 1, for half-rate interfaces the value is 2, and for quarter-rate interfaces the value is 4. |
| <code>DATA_RATE_RATIO</code> | The number of data bits transmitted per clock cycle. For single-data rate protocols this value is 1, and for double-data rate protocols this value is 2.   |
| <code>ADDR_RATE_RATIO</code> | The number of address bits transmitted per clock cycle. For single-data rate address protocols this value is 1, and for double-data rate address protocols this value is 2.                                  |

**Table 7. Memory Interface Parameters**

| Parameter Name                      | Description  |
|-------------------------------------|--|
| <code>MEM_IF_ADDR_WIDTH</code>      | The width of the address bus on the memory device(s).  |
| <code>MEM_IF_BANKADDR_WIDTH</code>  | The width of the bank address bus on the interface to the memory device(s). Typically, the $\log_2$ of the number of banks.                                |
| <code>MEM_IF_CS_WIDTH</code>        | The number of chip selects on the interface to the memory device(s).   |
| <code>MEM_IF_WRITE_DQS_WIDTH</code> | The number of DQS (or write clock) signals on the write interface. For example, the number of DQS groups.  |
| <code>MEM_IF_CLK_PAIR_COUNT</code>  | The number of CK/CK# pairs.  |
| <code>MEM_IF_DQ_WIDTH</code>        | The number of DQ signals on the interface to the memory device(s). For single-ended interfaces such as QDR II, this value is the number of D or Q signals. |
| <code>MEM_IF_DM_WIDTH</code>        | The number of data mask pins on the interface to the memory device(s).   |
| <code>MEM_IF_READ_DQS_WIDTH</code>  | The number of DQS signals on the read interface. For example, the number of DQS groups.  |

**Table 8. Derived AFI Parameters**

| Parameter Name                   | Derivation Equation   |
|----------------------------------|---|
| <code>AFI_ADDR_WIDTH</code>      | <code>MEM_IF_ADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO</code>     |
| <code>AFI_BANKADDR_WIDTH</code>  | <code>MEM_IF_BANKADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO</code> |
| <code>AFI_CONTROL_WIDTH</code>   | <code>AFI_RATE_RATIO * ADDR_RATE_RATIO</code>                         |
| <code>AFI_CS_WIDTH</code>        | <code>MEM_IF_CS_WIDTH * AFI_RATE_RATIO</code>                         |
| <code>AFI_DM_WIDTH</code>        | <code>MEM_IF_DM_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO</code>       |
| <code>AFI_DQ_WIDTH</code>        | <code>MEM_IF_DQ_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO</code>       |
| <code>AFI_WRITE_DQS_WIDTH</code> | <code>MEM_IF_WRITE_DQS_WIDTH * AFI_RATE_RATIO</code>                  |
| <code>AFI_LAT_WIDTH</code>       | 6   |

*continued...*



| Parameter Name     | Derivation Equation                                      |
|--------------------|--|
| AFI_RLAT_WIDTH     | AFI_LAT_WIDTH  |
| AFI_WLAT_WIDTH     | AFI_LAT_WIDTH * MEM_IF_WRITE_DQS_WIDTH                   |
| AFI_CLK_PAIR_COUNT | MEM_IF_CLK_PAIR_COUNT                                    |
| AFI_WRANK_WIDTH    | Number of ranks * MEM_IF_WRITE_DQS_WIDTH *AFI_RATE_RATIO |
| AFI_RRANK_WIDTH    | Number of ranks * MEM_IF_READ_DQS_WIDTH *AFI_RATE_RATIO  |

### 1.13.3. AFI Signals

The following tables list Altera PHY interface (AFI) signals grouped according to their functions.

In each table, the **Direction** column denotes the direction of the signal relative to the PHY. For example, a signal defined as an output passes out of the PHY to the controller. The AFI specification does not include any bidirectional signals.

Not all signals are used for all protocols.

#### 1.13.3.1. AFI Clock and Reset Signals

The AFI interface provides up to two clock signals and an asynchronous reset signal.

**Table 9. Clock and Reset Signals**

| Signal Name  | Direction | Width | Description  |
|--------------|-----------|-------|--|
| afi_clk      | Output    | 1     | Clock with which all data exchanged on the AFI bus is synchronized. In general, this clock is referred to as full-rate, half-rate, or quarter-rate, depending on the ratio between the frequency of this clock and the frequency of the memory device clock. |
| afi_half_clk | Output    | 1     | Clock signal that runs at half the speed of the afi_clk. The controller uses this signal when the half-rate bridge feature is in use. This signal is optional.   |
| afi_reset_n  | Output    | 1     | Asynchronous reset output signal. You must synchronize this signal to the clock domain in which you use it.  |

#### 1.13.3.2. AFI Address and Command Signals

The address and command signals for AFI 3.0 encode read/write/configuration commands to send to the memory device. The address and command signals are single-data rate signals.

**Table 10. Address and Command Signals**

| Signal Name | Direction | Width              | Description                                |
|-------------|-----------|--------------------|--|
| afi_ba      | Input     | AFI_BANKADDR_WIDTH | Bank address. (Not applicable for LPDDR3.) |
| afi_cke     | Input     | AFI_CLK_EN_WIDTH   | Clock enable.                              |

*continued...*



| Signal Name         | Direction | Width              | Description   |
|---------------------|-----------|--------------------|---|
| afi_cs_n            | Input     | AFI_CS_WIDTH       | Chip select signal. (The number of chip selects may not match the number of ranks; for example, RDIMMs and LRDIMMs require a minimum of 2 chip select signals for both single-rank and dual-rank configurations. Consult your memory device data sheet for information about chip select signal width.) (Matches the number of ranks for LPDDR3.) |
| afi_ras_n           | Input     | AFI_CONTROL_WIDTH  | RAS# (for DDR2 and DDR3 memory devices.)  |
| afi_we_n            | Input     | AFI_CONTROL_WIDTH  | WE# (for DDR2, DDR3, and RDRAM II memory devices.)  |
| afi_cas_n           | Input     | AFI_CONTROL_WIDTH  | CAS# (for DDR2 and DDR3 memory devices.)  |
| afi_ref_n           | Input     | AFI_CONTROL_WIDTH  | REF# (for RDRAM II memory devices.)   |
| afi_RST_n           | Input     | AFI_CONTROL_WIDTH  | RESET# (for DDR3 and DDR4 memory devices.)  |
| afi_odt             | Input     | AFI_CLK_EN_WIDTH   | On-die termination signal for DDR2, DDR3, and LPDDR3 memory devices. (Do not confuse this memory device signal with the FPGA's internal on-chip termination signal.)  |
| afi_mem_clk_disable | Input     | AFI_CLK_PAIR_COUNT | When this signal is asserted, mem_clk and mem_clk_n are disabled. This signal is used in low-power mode.  |
| afi_wps_n           | Output    | AFI_CS_WIDTH       | WPS (for QDR II/II+ memory devices.)  |
| afi_rps_n           | Output    | AFI_CS_WIDTH       | RPS (for QDR II/II+ memory devices.)  |

### 1.13.3.3. AFI Write Data Signals

Write Data Signals for AFI 3.0 control the data, data mask, and strobe signals passed to the memory device during write operations.

**Table 11. Write Data Signals**

| Signal Name   | Direction | Width               | Description  |
|---------------|-----------|---------------------|--|
| afi_dqs_burst | Input     | AFI_WRITE_DQS_WIDTH | Controls the enable on the strobe (DQS) pins for DDR2, DDR3, and LPDDR2 memory devices. When this signal is asserted, mem_dqs and mem_dqsn are driven. |

*continued...*



| Signal Name     | Direction | Width               | Description   |
|-----------------|-----------|---------------------|---|
|                 |           |                     | This signal must be asserted before afi_wdata_valid to implement the write preamble, and must be driven for the correct duration to generate a correctly timed mem_dqs signal.  |
| afi_wdata_valid | Input     | AFI_WRITE_DQS_WIDTH | Write data valid signal. This signal controls the output enable on the data and data mask pins.   |
| afi_wdata       | Input     | AFI_DQ_WIDTH        | Write data signal to send to the memory device at double-data rate. This signal controls the PHY's mem_dq output.   |
| afi_dm          | Input     | AFI_DM_WIDTH        | Data mask. This signal controls the PHY's mem_dm signal for DDR2, DDR3, LPDDR2 and RLDRAM II memory devices.)   |
| afi_bws_n       | Input     | AFI_DM_WIDTH        | Data mask. This signal controls the PHY's mem_bws_n signal for QDR II/II+ memory devices.   |
| afi_wrank       | Input     | AFI_WRANK_WIDTH     | Shadow register signal. Signal indicating the rank to which the controller is writing, so that the PHY can switch to the appropriate setting. Signal timing is identical to afi_dqs_burst; that is, afi_wrank must be asserted at the same time as afi_dqs_burst, and must be of the same duration. |

#### 1.13.3.4. AFI Read Data Signals

Read Data Signals for AFI 3.0 control the data sent from the memory device during read operations.

**Table 12. Read Data Signals**

| Signal Name       | Direction | Width          | Description  |
|-------------------|-----------|----------------|--|
| afi_rdata_en      | Input     | AFI_RATE_RATIO | Read data enable. Indicates that the memory controller is currently performing a read operation. This signal is held high only for cycles of relevant data (read data masking). If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2). |
| afi_rdata_en_full | Input     | AFI_RATE_RATIO | Read data enable full. Indicates that the memory controller is currently performing a read operation. This signal is held high for the entire read burst. If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2).                       |

*continued...*



| Signal Name     | Direction | Width           | Description   |
|-----------------|-----------|-----------------|---|
| afi_rdata       | Output    | AFI_DQ_WIDTH    | Read data from the memory device. This data is considered valid only when afi_rdata_valid is asserted by the PHY.   |
| afi_rdata_valid | Output    | AFI_RATE_RATIO  | Read data valid. When asserted, this signal indicates that the afi_rdata bus is valid. If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2).   |
| afi_rrank       | Input     | AFI_RRANK_WIDTH | Shadow register signal. Signal indicating the rank from which the controller is reading, so that the PHY can switch to the appropriate setting. Must be asserted at the same time as afi_rdata_en when issuing a read command, and once asserted, must remain unchanged until the controller issues a new read command to another rank. |

### 1.13.3.5. AFI Calibration Status Signals

The PHY instantiates a sequencer which calibrates the memory interface with the memory device and some internal components such as read FIFOs and valid FIFOs. The sequencer reports the results of the calibration process to the controller through the Calibration Status Signals in the AFI interface.

**Table 13. Calibration Status Signals**

| Signal Name  | Direction | Width          | Description   |
|--|-----------|----------------|---|
| afi_cal_success  | Output    | 1              | Asserted to indicate that calibration has completed successfully.   |
| afi_cal_fail   | Output    | 1              | Asserted to indicate that calibration has failed.   |
| afi_cal_req  | Input     | 1              | Effectively a synchronous reset for the sequencer. When this signal is asserted, the sequencer returns to the reset state; when this signal is released, a new calibration sequence begins.   |
| afi_wlat   | Output    | AFI_WLAT_WIDTH | The required write latency in afi_clk cycles, between address/command and write data being issued at the PHY/controller interface. The afi_wlat value can be different for different groups; each group's write latency can range from 0 to 63. If write latency is the same for all groups, only the lowest 6 bits are required. |
| afi_rlat<br><sup>(1)</sup>   | Output    | AFI_RLAT_WIDTH | The required read latency in afi_clk cycles between address/command and read data being returned to the PHY/controller interface. Values can range from 0 to 63.  |
| Note to Table:   |           |                |   |
| 1. The afi_rlat signal is not supported for PHY-only designs. Instead, you can sample the afi_rdata_valid signal to determine when valid read data is available. |           |                |   |



### 1.13.3.6. AFI Tracking Management Signals

When tracking management is enabled, the sequencer can take control over the AFI 3.0 interface at given intervals, and issue commands to the memory device to track the internal DQS Enable signal alignment to the DQS signal returning from the memory device. The tracking management portion of the AFI 3.0 interface provides a means for the sequencer and the controller to exchange handshake signals.

**Table 14. Tracking Management Signals**

| Signal Name          | Direction | Width           | Description   |
|----------------------|-----------|-----------------|---|
| afi_ctl_refresh_done | Input     | MEM_IF_CS_WIDTH | Handshaking signal from controller to tracking manager, indicating that a refresh has occurred and waiting for a response.                                  |
| afi_seq_busy         | Output    | MEM_IF_CS_WIDTH | Handshaking signal from sequencer to controller, indicating when DQS tracking is in progress.   |
| afi_ctl_long_idle    | Input     | MEM_IF_CS_WIDTH | Handshaking signal from controller to tracking manager, indicating that it has exited low power state without a periodic refresh, and waiting for response. |

## 1.14. Register Maps

The following table lists the overall register mapping for the DDR2, DDR3, and LPDDR2 SDRAM Controllers with UniPHY.

*Note:* Addresses shown in the table are 32-bit word addresses. If a byte-addressed master such as a Nios II processor accesses the CSR, it is necessary to multiply the addresses by four.

**Table 15. Register Map**

| Address                        | Description   |
|--------------------------------|---|
| <b>UniPHY Register Map</b>     |   |
| 0x001                          | Reserved.   |
| 0x004                          | UniPHY status register 0.                           |
| 0x005                          | UniPHY status register 1.                           |
| 0x006                          | UniPHY status register 2.                           |
| 0x007                          | UniPHY memory initialization parameters register 0. |
| <b>Controller Register Map</b> |   |
| 0x100                          | Reserved.   |
| 0x110                          | Controller status and configuration register.       |
| 0x120                          | Memory address size register 0.                     |
| 0x121                          | Memory address size register 1.                     |
| 0x122                          | Memory address size register 2.                     |
| 0x123                          | Memory timing parameters register 0.                |

*continued...*



| Address | Description                          |
|---------|--------------------------------------|
| 0x124   | Memory timing parameters register 1. |
| 0x125   | Memory timing parameters register 2. |
| 0x126   | Memory timing parameters register 3. |
| 0x130   | ECC control register.                |
| 0x131   | ECC status register.                 |
| 0x132   | ECC error address register.          |

### 1.14.1. UniPHY Register Map

The UniPHY register map allows you to control the memory components' mode register settings. The following table lists the register map for UniPHY.

**Note:** Addresses shown in the table are 32-bit word addresses. If a byte-addressed master such as a Nios II processor accesses the CSR, it is necessary to multiply the addresses by four.

**Table 16. UniPHY Register Map**

| Address | Bit   | Name               | Default | Access     | Description   |
|---------|-------|--------------------|---------|------------|---|
| 0x001   | 15:0  | Reserved.          | 0       | —          | Reserved for future use.  |
|         | 31:16 | Reserved.          | 0       | —          | Reserved for future use.  |
| 0x002   | 15:0  | Reserved.          | 0       | —          | Reserved for future use.  |
|         | 31:16 | Reserved.          | 0       | —          | Reserved for future use.  |
| 0x004   | 0     | SOFT_RESET         | —       | Write only | Initiate a soft reset of the interface. This bit is automatically deasserted after reset.   |
|         | 23:1  | Reserved.          | 0       | —          | Reserved for future use.  |
|         | 24    | AFI_CAL_SUCCESS    | —       | Read only  | Reports the value of the UniPHY afi_cal_success. Writing to this bit has no effect.   |
|         | 25    | AFI_CAL_FAIL       | —       | Read only  | Reports the value of the UniPHY afi_cal_fail. Writing to this bit has no effect.  |
|         | 26    | PLL_LOCKED         | —       | Read only  | Reports the PLL lock status.  |
|         | 31:27 | Reserved.          | 0       | —          | Reserved for future use.  |
|         | 7:0   | Reserved.          | 0       | —          | Reserved for future use.  |
| 0x005   | 15:8  | Reserved.          | 0       | —          | Reserved for future use.  |
|         | 23:16 | Reserved.          | 0       | —          | Reserved for future use.  |
|         | 31:24 | Reserved.          | 0       | —          | Reserved for future use.  |
|         | 7:0   | INIT FAILING STAGE | —       | Read only  | Initial failing error stage of calibration. Only applicable if AFI_CAL_FAIL=1.<br>0: None<br>1: Read Calibration - VFIFO<br>2: Write Calibration - Write Leveling |

*continued...*



| Address     | Bit   | Name                  | Default | Access    | Description   |
|-------------|-------|-----------------------|---------|-----------|---|
|             |       |                       |         |           | 3: Read Calibration - LFIFO Calibration<br>4: Write Calibration - Write Deskew<br>5: Unused<br>6: Refresh<br>7: Calibration Skipped<br>8: Calibration Aborted<br>9: Read Calibration - VFIFO After Writes   |
|             | 15:8  | INIT FAILING SUBSTAGE | —       | Read only | Initial failing error substage of calibration. Only applicable if AFI_CAL_FAIL=1.<br>If INIT FAILING STAGE = 1 or 9:<br>1: Read Calibration - Guaranteed read failure<br>2: Read Calibration - No working DQSen phase found<br>3: Read Calibration - Per-bit read deskew failure<br>If INIT FAILING STAGE = 2:<br>1: Write Calibration - No first working write leveling phase found<br>2: Write Calibration - No last working write leveling phase found<br>3: Write Calibration - Write leveling copy failure<br>If INIT FAILING STAGE = other, substages do not apply. |
|             | 23:16 | INIT FAILING GROUP    | —       | Read only | Initial failing error group of calibration. Only applicable if AFI_CAL_FAIL=1.<br>Returns failing DQ pin instead of failing group, if:<br>INIT FAILING STAGE=1 and INIT FAILING SUBSTAGE=3.<br>Or<br>INIT FAILING STAGE=4 and INIT FAILING SUBSTAGE=1.  |
|             | 31:24 | Reserved.             | 0       | —         | Reserved for future use.  |
| 0x007       | 31:0  | DQS_DETECT            | —       | Read only | Identifies if DQS edges have been identified for each of the groups. Each bit corresponds to one DQS group.   |
| 0x008(DDR2) | 1:0   | RTT_NOM               | —       | Read only | Rtt (nominal) setting of the DDR2 Extended Mode Register used during memory initialization.   |
|             | 31:2  | Reserved.             | 0       | —         | Reserved for future use.  |
| 0x008(DDR3) | 2:0   | RTT_NOM               | —       |           | Rtt (nominal) setting of the DDR3 MR1 mode register used during memory initialization.  |
|             | 4:3   | Reserved.             | 0       |           | Reserved for future use.  |
|             | 6:5   | ODS                   | —       |           | Output driver impedance control setting of the DDR3 MR1 mode register used during memory initialization.  |

continued...



| Address           | Bit   | Name      | Default | Access | Description   |
|-------------------|-------|-----------|---------|--------|---|
|                   | 8:7   | Reserved. | 0       |        | Reserved for future use.  |
|                   | 10:9  | RTT_WR    | —       |        | Rtt (writes) setting of the DDR3 MR2 mode register used during memory initialization. |
|                   | 31:11 | Reserved. | 0       |        | Reserved for future use.  |
| 0x008(LP<br>DDR2) | 3:0   | DS        |         |        | Driver impedance control for MR3 during initialization.                               |
|                   | 31:4  | Reserved. |         |        | Reserved for future use.  |

## 1.14.2. Controller Register Map

The controller register map allows you to control the memory controller settings.

*Note:* Dynamic reconfiguration is not currently supported.

For information on the controller register map, refer to *Controller Register Map*, in the *Functional Description—HPC II* chapter.

### Related Information

- [Soft Controller Register Map](#) on page 364
- [Hard Controller Register Map](#) on page 368

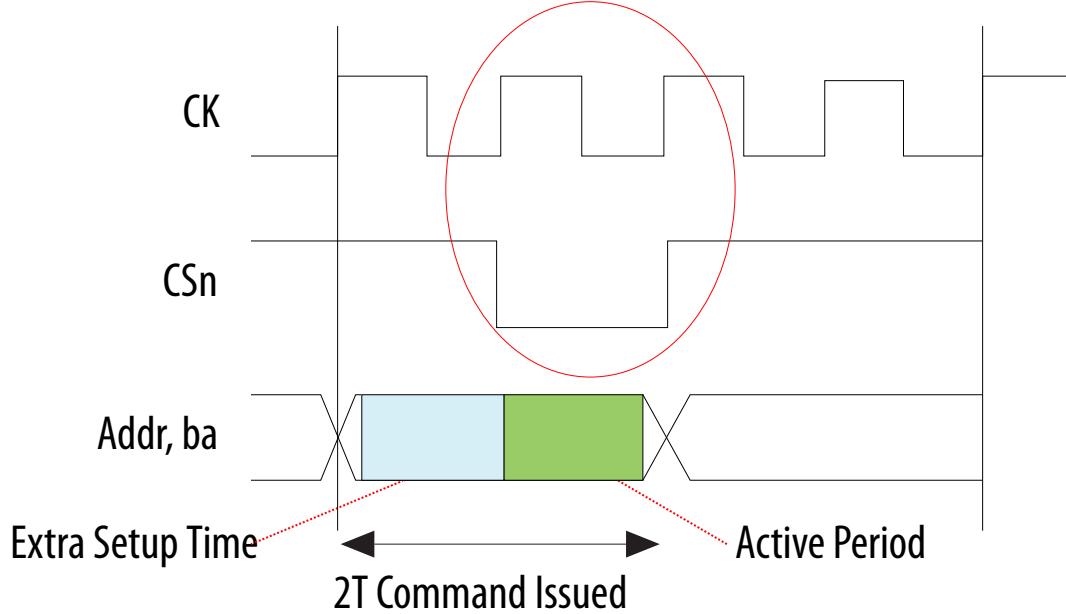
## 1.15. Ping Pong PHY

Ping Pong PHY is an implementation of UniPHY that allows two memory interfaces to share address and command buses through time multiplexing. Compared to having two independent interfaces, Ping Pong PHY uses fewer pins and less logic, while maintaining equivalent throughput.

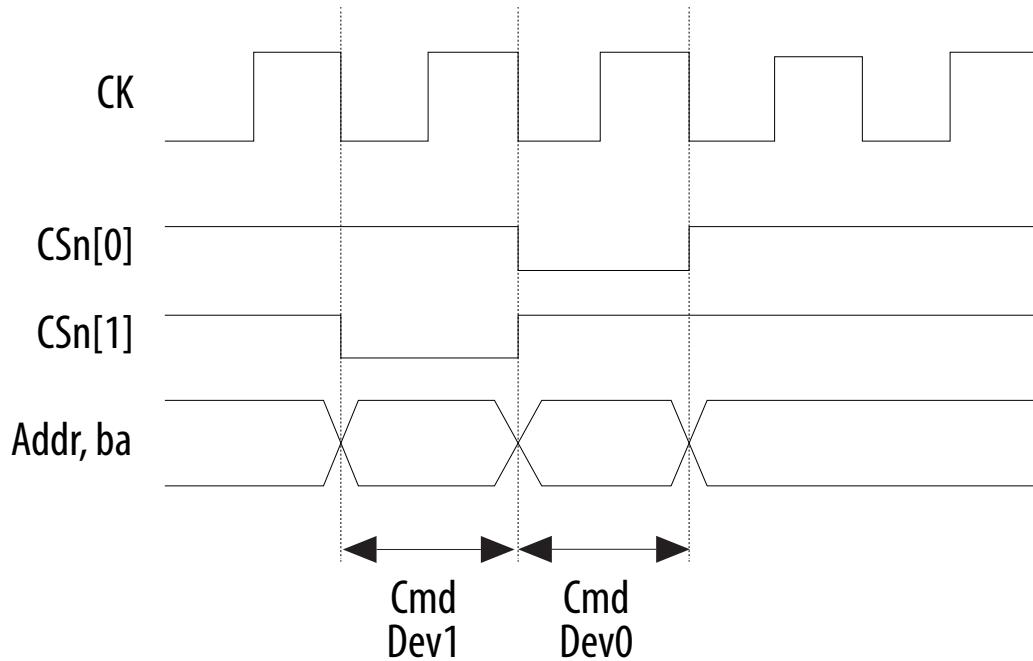
The Ping Pong PHY supports only quarter-rate configurations of the DDR3 protocol on Arria V GZ and Stratix V devices.

### 1.15.1. Ping Pong PHY Feature Description

In conventional UniPHY, the address and command buses of a DDR3 quarter-rate interface use 2T time—meaning that they are issued for two full-rate clock cycles, as illustrated below.

**Figure 19. 2T Command Timing**


With the Ping Pong PHY, address and command signals from two independent controllers are multiplexed onto shared buses by delaying one of the controller outputs by one full-rate clock cycle. The result is 1T timing, with a new command being issued on each full-rate clock cycle. The following figure shows address and command timing for the Ping Pong PHY.

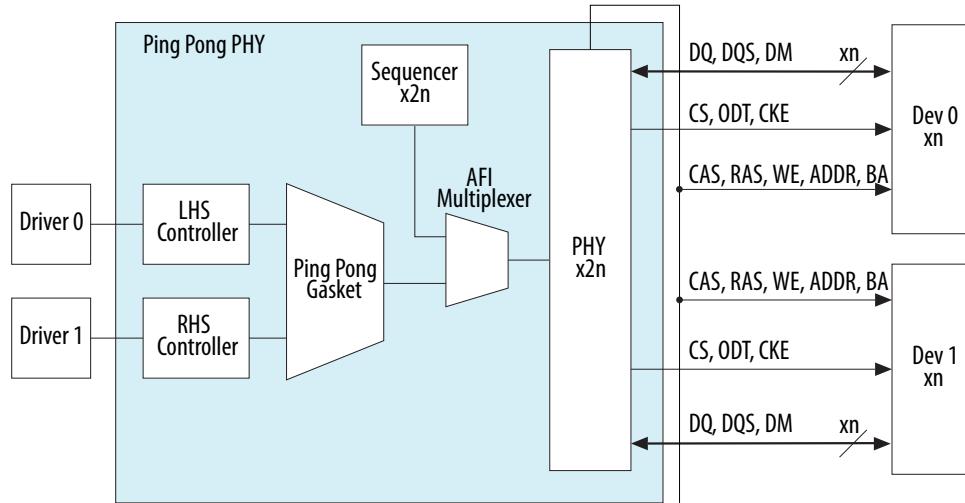
**Figure 20. 1T Command Timing Use by Ping Pong PHY**




## 1.15.2. Ping Pong PHY Architecture

The following figure shows a top-level block diagram of the Ping Pong PHY. Functionally, the IP looks like two independent memory interfaces. The two controller blocks are referred to as right-hand side (RHS) and left-hand side (LHS), respectively. A gasket block located between the controllers and the PHY merges the AFI signals. The PHY is double data width and supports both memory devices. The sequencer is the same as with regular UniPHY, and calibrates the entire double-width PHY.

**Figure 21. Ping Pong PHY Architecture**

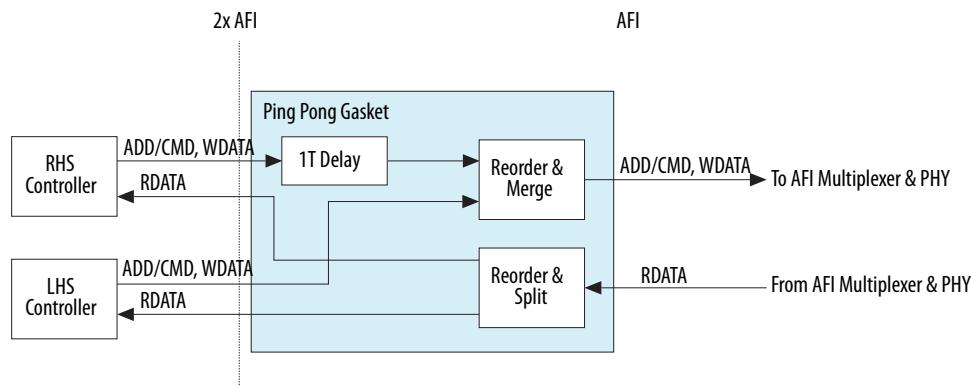


### 1.15.2.1. Ping Pong Gasket

The gasket delays and remaps quarter-rate signals so that they are correctly time-multiplexed at the full-rate PHY output. The gasket also merges address and command buses ahead of the PHY.

AFI interfaces at the input and output of the gasket provide compatibility with the PHY and with memory controllers.

**Figure 22. Ping Pong PHY Gasket Architecture**



The following table shows how the gasket processes key AFI signals.

**Table 17. Key AFI Signals Processed by Ping Pong PHY Gasket**

| Signal  | Direction(Width multiplier) | Description   | Gasket Conversions               |
|---|-----------------------------|---|----------------------------------|
| cas, ras, we, addr, ba                                  | Controller (1x) to PHY (1x) | Address and command buses shared between devices.   | Delay RHS by 1T; merge.          |
| cs, odt, cke  | Controller (1x) to PHY (2x) | Chip select, on-die termination, and clock enable, one per device.                              | Delay RHS by 1T; reorder, merge. |
| wdata, wdata_valid, dqs_burst, dm                       | Controller (1x) to PHY (2x) | Write datapath signals, one per device.   | Delay RHS by 1T; reorder, merge. |
| rdata_en_rd, rdata_en_rd_full                           | Controller (1x) to PHY (2x) | Read datapath enable signals indicating controller performing a read operation, one per device. | Delay RHS by 1T.                 |
| rdata_rdata_valid                                       | PHY (2x) to Controller (1x) | Read data, one per device.  | Reorder; split.                  |
| cal_fail, cal_success, seq_busy, wlat, rlat             | PHY (1x) to Controller (1x) | Calibration result, one per device.   | Pass through.                    |
| rst_n, mem_clk_disable, ctl_refresh_done, ctl_long_idle | Controller (1x) to PHY (1x) | Reset and DQS tracking signals, one per PHY.  | AND (&)                          |
| cal_req, init_req                                       | Controller (1x) to PHY (1x) | Controller to sequencer requests.   | OR ( )                           |
| wrank, rrank  | Controller (1x) to PHY (2x) | Shadow register support.  | Delay RHS by 1T; reorder; merge. |

### 1.15.2.2. Ping Pong PHY Calibration

The sequencer treats the Ping Pong PHY as a regular interface of double the width. For example, in the case of two x16 devices, the sequencer calibrates both devices together as a x32 interface. The sequencer chip select signal fans out to both devices so that they are treated as a single interface. The VFIFO calibration process is unchanged. For LFIFO calibration, the LFIFO buffer is duplicated for each interface and the worst-case read datapath delay of both interfaces is used.

### 1.15.3. Ping Pong PHY Operation

To use the Ping Pong PHY, proceed as described below.

1. Configure a single memory interface according to your requirements.
2. Select the **Enable Ping Pong PHY** option in the **Advanced PHY Options** section of the **PHY Settings** tab in the DDR3 parameter editor.

The Quartus Prime software then replicates the interface, resulting in two memory controllers and a shared PHY, with the gasket block inserted between the controllers and PHY. The system makes the necessary modifications to top-level component connections, as well as the PHY read and write datapaths, and the AFI mux, without further input from you.

## 1.16. Efficiency Monitor and Protocol Checker

The Efficiency Monitor and Protocol Checker allows measurement of traffic efficiency on the Avalon-MM bus between the controller and user logic, measures read latencies, and checks the legality of Avalon commands passed from the master. The Efficiency



Monitor and Protocol Checker is available with the DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY and the RLDRAM II Controller with UniPHY. The Efficiency Monitor and Protocol Checker is not available for QDR II and QDR II+ SRAM, or for the MAX 10 device family, or for Arria V or Cyclone V designs using the Hard Memory Controller.

### 1.16.1. Efficiency Monitor

The Efficiency Monitor reports read and write throughput on the controller input, by counting command transfers and wait times, and making that information available to the External Memory Interface Toolkit via an Avalon slave port. This information may be useful to you when experimenting with advanced controller settings, such as command look ahead depth and burst merging.

### 1.16.2. Protocol Checker

The Protocol Checker checks the legality of commands on the controller's input interface against the Intel Avalon interface specification, and sets a flag in a register on an Avalon slave port if an illegal command is detected.

### 1.16.3. Read Latency Counter

The Read Latency Counter measures the minimum and maximum wait times for read commands to be serviced on the Avalon bus. Each read command is time-stamped and placed into a FIFO buffer upon arrival, and latency is determined by comparing that timestamp to the current time when the first beat of the returned read data is provided back to the master.

### 1.16.4. Using the Efficiency Monitor and Protocol Checker

To include the Efficiency Monitor and Protocol Checker when you generate your IP core, proceed as described below.

1. On the **Diagnostics** tab in the parameter editor, turn on **Enable the Efficiency Monitor and Protocol Checker on the Controller Avalon Interface**.
2. To see the results of the data compiled by the Efficiency Monitor and Protocol Checker, use the External Memory Interface Toolkit.

For information on the External Memory Interface Toolkit, refer to *External Memory Interface Debug Toolkit*, in section 2 of this volume. For information about the Avalon interface, refer to *Avalon Interface Specifications*.

#### Related Information

- [Avalon Interface Specifications](#)
- [External Memory Interface Debug Toolkit](#) on page 471



### 1.16.5. Avalon CSR Slave and JTAG Memory Map

The following table lists the memory map of registers inside the Efficiency Monitor and Protocol Checker. This information is only of interest if you want to communicate directly with the Efficiency Monitor and Protocol Checker without using the External Memory Interface Toolkit. This CSR map is not part of the UniPHY CSR map.

Prior to reading the data in the CSR, you must issue a read command to address 0x01 to take a snapshot of the current data.

**Table 18. Avalon CSR Slave and JTAG Memory Map**

| Address | Bit   | Name                                 | Default | Access     | Description   |
|---------|-------|--------------------------------------|---------|------------|---|
| 0x01    | 31:0  | Reserved                             | 0       | Read Only  | Used internally by EMIF Toolkit to identify Efficiency Monitor type. This address must be read prior to reading the other CSR contents. |
| 0x02    | 31:0  | Reserved                             | 0       | —          | Used internally by EMIF Toolkit to identify Efficiency Monitor version.   |
| 0x08    | 0     | Efficiency Monitor reset             | —       | Write only | Write a 0 to reset.   |
|         | 7:1   | Reserved                             | —       | —          | Reserved for future use.  |
|         | 8     | Protocol Checker reset               | —       | Write only | Write a 0 to reset.   |
|         | 15:9  | Reserved                             | —       | —          | Reserved for future use.  |
|         | 16    | Start/stop Efficiency Monitor        | —       | Read/Write | Starting and stopping statistics gathering.   |
|         | 23:17 | Reserved                             | —       | —          | Reserved for future use.  |
|         | 31:24 | Efficiency Monitor status            | —       | Read Only  | bit 0: Efficiency Monitor stopped<br>bit 1: Waiting for start of pattern<br>bit 2: Running<br>bit 3: Counter saturation                 |
| 0x10    | 15:0  | Efficiency Monitor address width     | —       | Read Only  | Address width of the Efficiency Monitor.  |
|         | 31:16 | Efficiency Monitor data width        | —       | Read Only  | Data Width of the Efficiency Monitor.   |
| 0x11    | 15:0  | Efficiency Monitor byte enable       | —       | Read Only  | Byte enable width of the Efficiency Monitor.  |
|         | 31:16 | Efficiency Monitor burst count width | —       | Read Only  | Burst count width of the Efficiency Monitor.  |
| 0x14    | 31:0  | Cycle counter                        | —       | Read Only  | Clock cycle counter for the Efficiency Monitor. Lists the number of clock cycles elapsed before the Efficiency Monitor stopped.         |
| 0x18    | 31:0  | Transfer counter                     | —       | Read Only  | Counts any read or write data transfer cycle.   |
| 0x1C    | 31:0  | Write counter                        | —       | Read Only  | Counts write requests, including those during bursts.   |
| 0x20    | 31:0  | Read counter                         | —       | Read Only  | Counts read requests.   |
| 0x24    | 31:0  | Readtotal counter                    | —       | Read Only  | Counts read requests (total burst requests).  |

*continued...*



| Address | Bit  | Name                          | Default | Access    | Description   |
|---------|------|-------------------------------|---------|-----------|---|
| 0x28    | 31:0 | NTC waitrequest counter       | —       | Read Only | Counts Non Transfer Cycles (NTC) due to slave wait request high.                              |
| 0x2C    | 31:0 | NTC noreaddatavalid counter   | —       | Read Only | Counts Non Transfer Cycles (NTC) due to slave not having read data.                           |
| 0x30    | 31:0 | NTC master write idle counter | —       | Read Only | Counts Non Transfer Cycles (NTC) due to master not issuing command, or pause in write burst.  |
| 0x34    | 31:0 | NTC master idle counter       | —       | Read Only | Counts Non Transfer Cycles (NTC) due to master not issuing command anytime.                   |
| 0x40    | 31:0 | Read latency min              | —       | Read Only | The lowest read latency value.  |
| 0x44    | 31:0 | Read latency max              | —       | Read Only | The highest read latency value.   |
| 0x48    | 31:0 | Read latency total [31:0]     | —       | Read Only | The lower 32 bits of the total read latency.  |
| 0x49    | 31:0 | Read latency total [63:32]    | —       | Read Only | The upper 32 bits of the total read latency.  |
| 0x50    | 7:0  | Illegal command               | —       | Read Only | Bits used to indicate which illegal command has occurred. Each bit represents a unique error. |
|         | 31:8 | Reserved                      | —       | —         | Reserved for future use.  |

## 1.17. UniPHY Calibration Stages

The DDR2, DDR3, and LPDDR2 SDRAM, QDR II and QDR II+ SRAM, and RLDRAM II Controllers with UniPHY, and the RLDRAM 3 PHY-only IP, go through several stages of calibration. Calibration information is useful in debugging calibration failures.

The section includes an overview of calibration, explanation of the calibration stages, and a list of generated calibration signals. The information in this section applies only to the Nios II-based sequencer used in the DDR2, DDR3, and LPDDR2 SDRAM Controllers with UniPHY versions 10.0 and later, and, optionally, in the QDR II and QDR II+ SRAM and RLDRAM II Controllers with UniPHY version 11.0 and later, and the RLDRAM 3 PHY-only IP. The information in this section applies to the Arria II GZ, Arria V, Arria V GZ, Cyclone V, Stratix III, Stratix IV, and Stratix V device families.

**Note:** For QDR II and QDR II+ SRAM and RLDRAM II Controllers with UniPHY version 11.0 and later, you have the option to select either the RTL-based sequencer or the Nios II-based sequencer. Generally, choose the RTL-based sequencer when area is the major consideration, and choose the Nios II-based sequencer when performance is the major consideration.

**Note:** For RLDRAM 3, write leveling is not performed. The sequencer does not attempt to optimize margin for the tCKDK timing requirement.

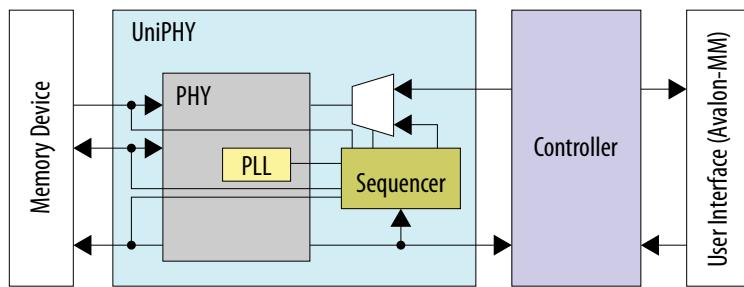
### 1.17.1. Calibration Overview

Calibration configures the memory interface (PHY and I/Os) so that data can pass reliably to and from memory.

The sequencer illustrated in the figure below calibrates the PHY and the I/Os. To correctly transmit data between a memory device and the FPGA at high speed, the data must be center-aligned with the data clock.

Calibration also determines the delay settings needed to center-align the various data signals with respect to their clocks. I/O delay chains implement the required delays in accordance with the computed alignments. The Nios II-based sequencer performs two major tasks: FIFO buffer calibration and I/O calibration. FIFO buffer calibration adjusts FIFO lengths and I/O calibration adjusts any delay chain and phase settings to center-align data signals with respect to clock signals for both reads and writes. When the calibration process completes, the sequencer shuts off and passes control to the memory controller.

**Figure 23. Sequencer in Memory Interface Logic**



### 1.17.2. Calibration Stages

The calibration process begins when the PHY reset signal deasserts and the PLL and DLL lock.

The following stages of calibration take place:

1. Read calibration part one—DQS enable calibration (only for DDR2 and DDR3 SDRAM Controllers with UniPHY) and DQ/DQS centering
2. Write calibration part one—Leveling
3. Write calibration part two—DQ/DQS centering
4. Read calibration part two—Read latency minimization

*Note:*

For multirank calibration, the sequencer transmits every read and write command to each rank in sequence. Each read and write test is successful only if all ranks pass the test. The sequencer calibrates to the intersection of all ranks.

The calibration process assumes the following conditions; if either of these conditions is not true, calibration likely fails in its early stages:

- The address and command paths must be functional; calibration does not tune the address and command paths. (The Quartus Prime software fully analyzes the timing for the address and command paths, and the slack report is accurate, assuming the correct board timing parameters.)
- At least one bit per group must work before running per-bit-deskew calibration. (This assumption requires that DQ-to-DQS skews be within the recommended 20 ps.)



### 1.17.3. Memory Initialization

The memory is powered up according to protocol initialization specifications. All ranks power up simultaneously. Once powered, the device is ready to receive mode register load commands. This part of initialization occurs separately for each rank. The sequencer issues mode register set commands on a per-chip-select basis and initializes the memory to the user-specified settings.

### 1.17.4. Stage 1: Read Calibration Part One—DQS Enable Calibration and DQ/DQS Centering

Read calibration occurs in two parts. Part one is DQS enable calibration with DQ/DQS centering, which happens during stage 1 of the overall calibration process; part two is read latency minimization, which happens during stage 4 of the overall calibration process.

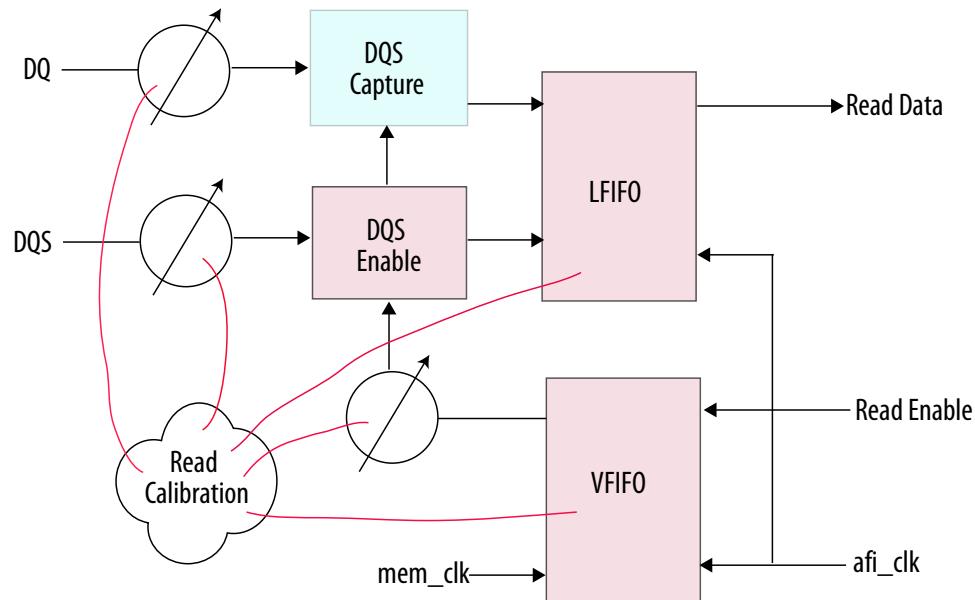
The objectives of DQS enable calibration and DQ/DQS centering are as follows:

- To calculate when the read data is received after a read command is issued to setup the Data Valid Prediction FIFO (VFIFO) cycle
- To align the input data (DQ) with respect to the clock (DQS) to maximize the read margins (DDR2 and DDR3 only)

DQS enable calibration and DQ/DQS centering consists of the following actions:

- Guaranteed Write
- DQS Enable Calibration
- DQ/DQS Centering

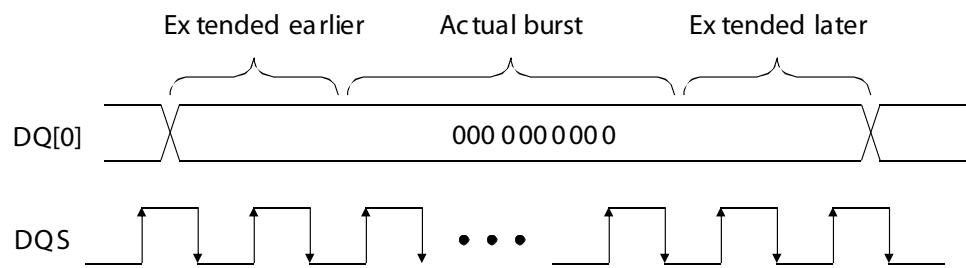
The following figure illustrates the components in the read data path that the sequencer calibrates in this stage. (The round knobs in the figure represent configurable hardware over which the sequencer has control.)

**Figure 24. Read Data Path Calibration Model**


#### 1.17.4.1. Guaranteed Write

Because initially no communication can be reliably performed with the memory device, the sequencer uses a guaranteed write mechanism to write data into the memory device. (For the QDR II protocol, guaranteed write is not necessary, a simple write mechanism is sufficient.)

The guaranteed write is a write command issued with all data pins, all address and bank pins, and all command pins (except chip select) held constant. The sequencer begins toggling DQS well before the expected latch time at memory and continues to toggle DQS well after the expected latch time at memory. DQ-to-DQS relationship is not a factor at this stage because DQ is held constant.

**Figure 25. Guaranteed Write of Zeros**




The guaranteed write consists of a series of back-to-back writes to alternating columns and banks. For example, for DQ[0] for the DDR3 protocol, the guaranteed write performs the following operations:

- Writes a full burst of zeros to bank 0, column 0
- Writes a full burst of zeros to bank 0, column 1
- Writes a full burst of ones to bank 3, column 0
- Writes a full burst of ones to bank 3, column 1

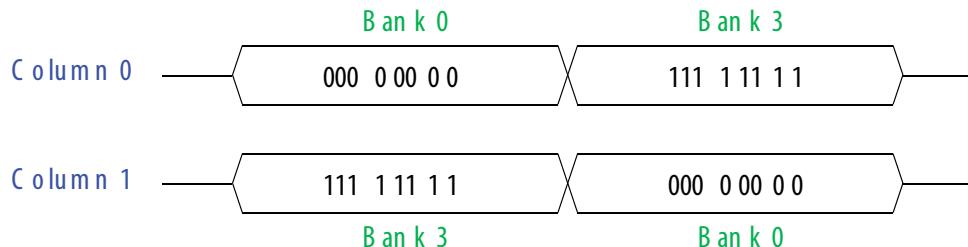
(Different protocols may use different combinations of banks and columns.)

The guaranteed write is followed by back-to-back read operations at alternating banks, effectively producing a stream of zeros followed by a stream of ones, or vice versa. The sequencer uses the zero-to-one and one-to-zero transitions in between the two bursts to identify a correct read operation, as shown in the figure below.

Although the approach described above for pin DQ[0] would work by writing the same pattern to all DQ pins, it is more effective and robust to write (and read) alternating ones and zeros to alternating DQ bits. The value of the DQ bit is still constant across the burst, and the back-to-back read mechanism works exactly as described above, except that odd DQ bits have ones instead of zeros, or vice versa.

The guaranteed write does not ensure a correct DQS-to-memory clock alignment at the memory device—DQS-to-memory clock alignment is performed later, in stage 2 of the calibration process. However, the process of guaranteed write followed by read calibration is repeated several times for different DQS-to-memory clock alignments, to ensure at least one correct alignment is found.

**Figure 26. Back to Back Reads on Pin DQ[0]**



#### 1.17.4.2. DQS Enable Calibration

DQS enable calibration ensures reliable capture of the DQ signal without glitches on the DQS line. At this point LFIFO is set to its maximum value to guarantee a reliable read from read capture registers to the core. Read latency is minimized later.

**Note:** The full DQS enable calibration is applicable only for DDR2 and DDR3 protocols; QDR II and RLDRAM protocols use only the VFIFO-based cycle-level calibration, described below.

**Note:** Delay and phase values used in this section are examples, for illustrative purposes. Your exact values may vary depending on device and configuration.

DQS enable calibration controls the timing of the enable signal using 3 independent controls: a cycle-based control (the VFIFO), a phase control, and a delay control. The VFIFO selects the cycle by shifting the controller-generated read data enable signal,

`rdata_en`, by a number of full-rate clock cycles. The phase is controlled using the DLL, while the delays are adjusted using a sequence of individual delay taps. The resolution of the phase and delay controls varies with family and configuration, but is approximately 45° for the phase, and between 10 and 50 picoseconds for the delays.

The sequencer finds the two edges of the DQS enable window by searching the space of cycles, phases, and delays (an exhaustive search can usually be avoided by initially assuming the window is at least one phase wide). During the search, to test the current settings, the sequencer issues back-to-back reads from column 0 of bank 0 and bank 3, and column 1 of bank 0 and bank 3, as shown in the preceding figure. Two full bursts are read and compared with the reference data for each phase and delay setting.

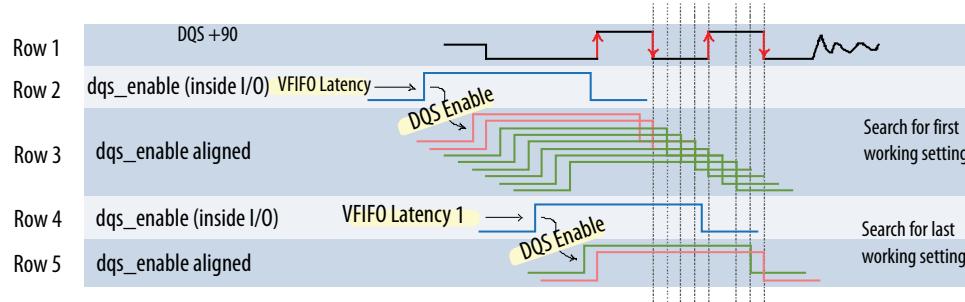
Once the sequencer identifies the two edges of the window, it center-aligns the falling edge of the DQS enable signal within the window. At this point, per-bit deskew has not yet been performed, therefore not all bits are expected to pass the read test; however, for read calibration to succeed, at least one bit per group must pass the read test.

The following figure shows the DQS and DQS enable signal relationship. The goal of DQS enable calibration is to find settings that satisfy the following conditions:

- The DQS enable signal rises before the first rising edge of DQS.
- The DQS enable signal is at one after the second-last falling edge of DQS.
- The DQS enable signal falls before the last falling edge of DQS.

The ideal position for the falling edge of the DQS enable signal is centered between the second-last and last falling edges of DQS.

**Figure 27. DQS and DQS Enable Signal Relationships**



The following points describe each row of the above figure:

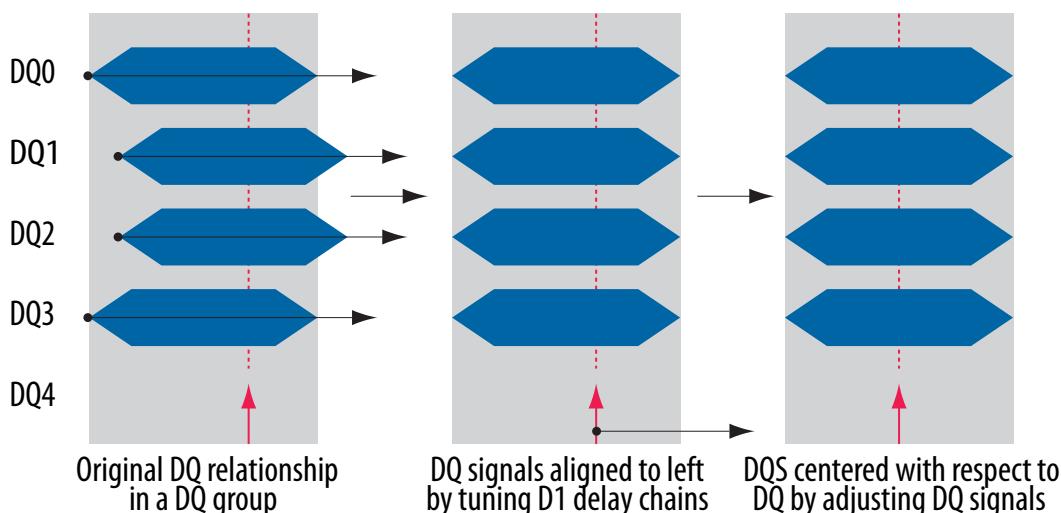
- Row 1 shows the DQS signal shifted by 90° to center-align it to the DQ data.
- Row 2 shows the raw DQS enable signal from the VFIFO.
- Row 3 shows the effect of sweeping DQS enable phases. The first two settings (shown in red) fail to properly gate the DQS signal because the enable signal turns off before the second-last falling edge of DQS. The next six settings (shown in green) gate the DQS signal successfully, with the DQS signal covering DQS from the first rising edge to the second-last falling edge.
- Row 4 shows the raw DQS enable signal from the VFIFO, increased by one clock cycle relative to Row 2.
- Row 5 shows the effect of sweeping DQS enable, beginning from the initial DQS enable of Row 4. The first setting (shown in green) successfully gates DQS, with the signal covering DQS from the first rising edge to the second-last falling edge. The second signal (shown in red), does not gate DQS successfully because the enable signal extends past the last falling edge of DQS. Any further adjustment would show the same failure.

#### 1.17.4.3. Centering DQ/DQS

The centering DQ/DQS stage attempts to align DQ and DQS signals on reads within a group. Each DQ signal within a DQS group might be skewed and consequently arrive at the FPGA at a different time. At this point, the sequencer sweeps each DQ signal in a DQ group to align them, by adjusting DQ input delay chains (D1).

The following figure illustrates a four DQ/DQS group per-bit-deskew and centering.

**Figure 28. Per-bit Deskew**



To align and center DQ and DQS, the sequencer finds the right edge of DQ signals with respect to DQS by sweeping DQ signals within a DQ group to the right until a failure occurs. In the above figure, DQ0 and DQ3 fail after six taps to the right; DQ1 and DQ2 fail after 5 taps to the right. To align the DQ signals, DQ0 and DQ3 are shifted to the right by 1 tap.

To find the center of DVW, the DQS signal is shifted to the right until a failure occurs. In the above figure, a failure occurs after 3 taps, meaning that there are 5 taps to the right edge and 3 taps to the left edge. To center-align DQ and DQS, the sequencer shifts the aligned DQ signal by 1 more tap to the right.

**Note:** The sequencer does not adjust DQS directly; instead, the sequencer center-aligns DQS with respect to DQ by delaying the DQ signals.

### 1.17.5. Stage 2: Write Calibration Part One

The objectives of the write calibration stage are to align DQS to the memory clock at each memory device, and to compensate for address, command, and memory clock skew at each memory device. This stage is important because the address, command, and clock signals for each memory component arrive at different times.

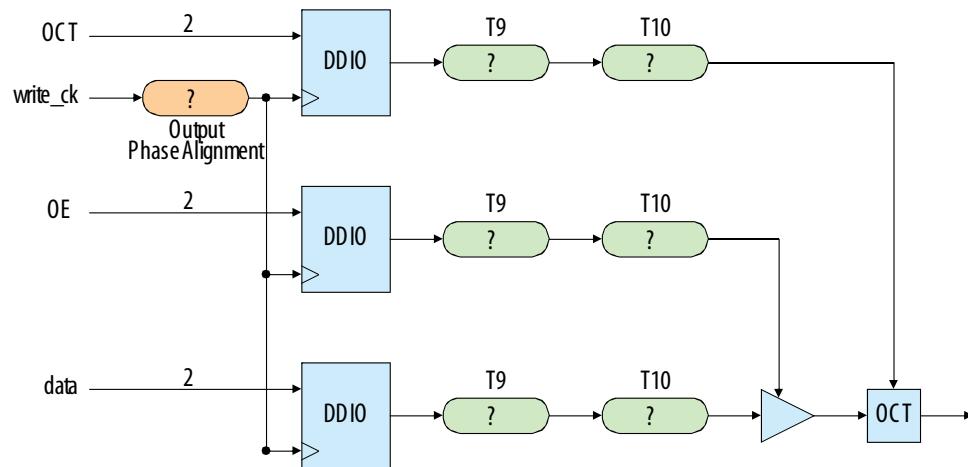
**Note:** This stage applies only to DDR2, DDR3, LPDDR2, and RLDRAM II protocols; it does not apply to the QDR II and QDR II+ protocols.

Memory clock signals and DQ/DM and DQS signals have specific relationships mandated by the memory device. The PHY must ensure that these relationships are met by skewing DQ/DM and DQS signals. The relationships between DQ/DM and DQS and memory clock signals must meet the tDQSS, tDSS, and tDSH timing constraints.

The sequencer calibrates the write data path using a variety of random burst patterns to compensate for the jitter on the output data path. Simple write patterns are insufficient to ensure a reliable write operation because they might cause imprecise DQS-to-CK alignments, depending on the actual capture circuitry on a memory device. The write patterns in the write leveling stage have a burst length of 8, and are generated by a linear feedback shift register in the form of a pseudo-random binary sequence.

The write data path architecture is the same for DQ, DM, and DQS pins. The following figure illustrates the write data path for a DQ signal. The phase coming out of the Output Phase Alignment block can be set to different values to center-align DQS with respect to DQ, and it is the same for data, OE, and OCT of a given output.

**Figure 29. Write Data Path**





In write leveling, the sequencer performs write operations with different delay and phase settings, followed by a read. The sequencer can implement any phase shift between 0° and 720° (depending on device and configuration). The sequencer uses the Output Phase Alignment for coarse delays and T9 and T10 for fine delays; T9 has 15 taps of 50 ps each, and T10 has 7 taps of 50 ps each.

The DQS signal phase is held at +90° with respect to DQ signal phase (Stratix IV example).

**Note:** Coarse delays are called *phases*, and fine delays are called *delays*; phases are process, voltage, and temperature (PVT) compensated, delays are not (depending on family).

For 28 nm devices:

- I/O delay chains are not PVT compensated.
- DQS input delay chain is PVT compensated.
- Leveling delay chains are PVT compensated (does not apply to Arria V or Cyclone V devices).
- T11 delay chain for postamble gating has PVT and nonPVT compensated modes, but the PVT compensated mode is not used.

**Note:** Delay and phase values used in this section are examples, for illustrative purposes. Your exact values may vary depending on device and configuration.

The sequencer writes and reads back several burst-length-8 patterns. Because the sequencer has not performed per-bit deskew on the write data path, not all bits are expected to pass the write test. However, for write calibration to succeed, at least one bit per group must pass the write test. The test begins by shifting the DQ/DQS phase until the first write operation completes successfully. The DQ/DQS signals are then delayed to the left by D5 and D6 to find the left edge for that working phase. Then DQ/DQS phase continues the shift to find the last working phase. For the last working phase, DQ/DQS is delayed in 50 ps steps to find the right edge of the last working phase.

The sequencer sweeps through all possible phase and delay settings for each DQ group where the data read back is correct, to define a window within which the PHY can reliably perform write operations. The sequencer picks the closest value to the center of that window as the phase/delay setting for the write data path.

### 1.17.6. Stage 3: Write Calibration Part Two—DQ/DQS Centering

The process of DQ/DQS centering in write calibration is similar to that performed in read calibration, except that write calibration is performed on the output path, using D5 and D6 delay chains.

### 1.17.7. Stage 4: Read Calibration Part Two—Read Latency Minimization

At this stage of calibration the sequencer adjusts LFIFO latency to determine the minimum read latency that guarantees correct reads.



### Read Latency Tuning

In general, DQ signals from different DQ groups may arrive at the FPGA in a staggered fashion. In a DIMM or multiple memory device system, the DQ/DQS signals from the first memory device arrive sooner, while the DQ/DQS signals from the last memory device arrive the latest at the FPGA.

LFIFO transfers data from the capture registers in IOE to the core and aligns read data to the AFI clock. Up to this point in the calibration process, the read latency has been a maximum value set initially by LFIFO; now, the sequencer progressively lowers the read latency until the data can no longer be transferred reliably. The sequencer then increases the latency by one cycle to return to a working value and adds an additional cycle of margin to assure reliable reads.

### 1.17.8. Calibration Signals

The following table lists signals produced by the calibration process.

**Table 19. Calibration Signals**

| Signal          | Description                                |
|-----------------|--|
| afi_cal_fail    | Asserts high if calibration fails.         |
| afi_cal_success | Asserts high if calibration is successful. |

### 1.17.9. Calibration Time

The time needed for calibration varies, depending on many factors including the interface width, the number of ranks, frequency, board layout, and difficulty of calibration. In general, designs using the Nios II-based sequencer will take longer to calibrate than designs using the RTL-based sequencer.

The following table lists approximate typical and maximum calibration times for various protocols.

**Table 20. Approximate Calibration Times**

| Protocol   | Typical Calibration Time | Maximum Calibration Time  |
|--|--------------------------|---|
| DDR2, DDR3, LPDDR2, RLDRAM 3                         | 50-250 ms                | Can take several minutes if the interface is difficult to calibrate, or if calibration initially fails and exhausts multiple retries. |
| QDR II/II+, RLDRAM II (with Nios II-based sequencer) | 50-100 ms                | Can take several minutes if the interface is difficult to calibrate, or if calibration initially fails and exhausts multiple retries. |
| QDR II/II+, RLDRAM II (with RTL-based sequencer)     | <5 ms                    | <5 ms   |

### 1.18. Document Revision History

| Date                | Version    | Changes  |
|---------------------|------------|--|
| May 2017            | 2017.05.08 | Rebranded as Intel.  |
| October 2016        | 2016.10.31 | <ul style="list-style-type: none"><li>Changed Note 3 to the <i>Word-Aligned Writes</i> figure, in the <i>PHY-to-Controller Interfaces</i> topic.</li></ul> |
| <i>continued...</i> |            |  |



| Date          | Version    | Changes   |
|---------------|------------|---|
| May 2016      | 2016.05.02 | <ul style="list-style-type: none"> <li>Added statement that Efficiency Monitor and Protocol checker is not available for QDR II and QDR II+ SRAM, or for the MAX 10 device family, or for Arria V or Cyclone V designs using the Hard Memory Controller, to <i>Efficiency Monitor and Protocol Checker</i> topic.</li> </ul>  |
| November 2015 | 2015.11.02 | <ul style="list-style-type: none"> <li>Replaced the <i>AFI 4.0 Specification</i> with the <i>AFI 3.0 Specification</i>.</li> <li>Replaced instances of <i>Quartus II</i> with <i>Quartus Prime</i>.</li> </ul>  |
| May 2015      | 2015.05.04 | Maintenance release.  |
| December 2014 | 2014.12.15 | <ul style="list-style-type: none"> <li>Added several parameters to the <i>AFI Specification</i> section:           <ul style="list-style-type: none"> <li>— MEM_IF_BANKGROUP_WIDTH</li> <li>— MEM_IF_C_WIDTH</li> <li>— MEM_IF_CKE_WIDTH</li> <li>— MEM_IF_ODT_WIDTH</li> <li>— AFI_BANKGROUP_WIDTH</li> <li>— AFI_C_WIDTH</li> <li>— AFI_CKE_WIDTH</li> <li>— AFI_ODT_WIDTH</li> </ul> </li> <li>Added several signals to the <i>AFI Specification</i> section:           <ul style="list-style-type: none"> <li>— afi_addr</li> <li>— afi_bg</li> <li>— afi_c_n</li> <li>— afi_rw_n</li> <li>— afi_act_n</li> <li>— afi_par</li> <li>— afi_alert_n</li> <li>— afi_ainv</li> <li>— afi_dinv</li> </ul> </li> <li>Changed the <i>Width</i> information for several signals in the <i>AFI Specification</i> section:           <ul style="list-style-type: none"> <li>— afi_dqs_burst</li> <li>— afi_wdata_valid</li> <li>— afi_stl_refresh_done</li> <li>— afi_seq_busy</li> <li>— afi_ctl_long_idle</li> </ul> </li> </ul> |
| August 2014   | 2014.08.15 | <ul style="list-style-type: none"> <li>Added note about 32-bit word addresses to <i>Register Maps</i> and <i>UniPHY Register Map</i> tables.</li> <li>Changed register map information for address 0x004, bit 26, in <i>UniPHY Register Map</i> table.</li> </ul>   |
| December 2013 | 2013.12.16 | <ul style="list-style-type: none"> <li>Removed references to HardCopy.</li> <li>Removed <i>DLL Offset Control Block</i>.</li> <li>Removed references to SOPC Builder.</li> <li>Increased minimum recommended pulse width for global_reset_n signal to 100ns.</li> <li>Corrected terminology inconsistency.</li> <li>Added information explaining PVT compensation.</li> <li>Added quarter-rate information to <i>PHY-to-Controller Interfaces</i> section.</li> <li>Expanded descriptions of INIT FAILING STAGE and INIT FAILING SUBSTAGE in UniPHY Register map.</li> <li>Added footnote about afi_rlat signal to <i>Calibration Status Signals</i> table.</li> </ul>  |
| November 2012 | 3.1        | <ul style="list-style-type: none"> <li>Moved <i>Controller Register Map</i> to <i>Functional Description—HPC II Controller</i> chapter.</li> <li>Updated Sequencer States information in Table 1-2.</li> <li>Enhanced <i>Using a Custom Controller</i> information.</li> <li>Enhanced <i>Tracking Manager</i> information.</li> </ul>   |

**continued...**



| Date          | Version | Changes  |
|---------------|---------|--|
|               |         | <ul style="list-style-type: none"><li>Added Ping Pong PHY information.</li><li>Added RLDRAM 3 support.</li><li>Added LRDIMM support.</li><li>Added Arria V GZ support.</li></ul>   |
| June 2012     | 3.0     | <ul style="list-style-type: none"><li>Added <i>Shadow Registers</i> section.</li><li>Added LPDDR2 support.</li><li>Added new AFI signals.</li><li>Added <i>Calibration Time</i> section.</li><li>Added Feedback icon.</li></ul>  |
| November 2011 | 2.1     | <ul style="list-style-type: none"><li>Consolidated UniPHY information from <i>11.0 DDR2 and DDR3 SDRAM Controller with UniPHY User Guide</i>, <i>QDR II and QDR II+ SRAM Controller with UniPHY User Guide</i>, and <i>RLDRAM II Controller with UniPHY IP User Guide</i>.</li><li>Revised <i>Reset and Clock Generation</i> and <i>Dedicated Clock Networks</i> sections.</li><li>Revised Figure 1–3 and Figure 1–5.</li><li>Added Tracking Manager to <i>Sequencer</i> section.</li><li>Revised <i>Interfaces</i> section for DLL, PLL, and OCT sharing interfaces.</li><li>Revised <i>Using a Custom Controller</i> section.</li><li>Added <i>UniPHY Calibration Stages</i> section; reordered stages 3 and 4, removed stage 5.</li></ul> |



## 2. Functional Description—Intel Stratix® 10 EMIF IP

Intel Stratix® 10 devices can interface with external memory devices clocking at frequencies of up to 1.3 GHz. The external memory interface IP component for Stratix 10 devices provides a single parameter editor for creating external memory interfaces, regardless of memory protocol. Unlike earlier EMIF solutions which used protocol-specific parameter editors to create memory interfaces via a complex RTL generation method, the Stratix 10 EMIF solution captures the protocol-specific hardened EMIF logic of the Stratix 10 device together with more generic soft logic.

The Stratix 10 EMIF solution is designed with the following implementations in mind:

### **Hard Memory Controller and Hard PHY**

This implementation provides a complete external memory interface, based on the hard memory controller and hard PHY that are part of the Stratix 10 silicon. An Avalon-MM interface is available for integration with user logic.

### **Soft Memory Controller and Hard PHY**

This implementation provides a complete external memory interface, using an Intel-provided soft-logic-based memory controller and the hard PHY that is part of the Stratix 10 silicon. An Avalon-MM interface is available for integration with user logic.

### **Custom Memory Controller and Hard PHY (PHY only)**

This implementation provides access to the Altera PHY interface (AFI), to allow use of a custom or third-party memory controller with the hard PHY that is part of the Stratix 10 silicon. Because only the PHY component is provided by Intel, this configuration is also known as *PHY only*.

### 2.1. Stratix 10 Supported Memory Protocols

The following table lists the external memory protocols supported by Stratix 10 devices.

**Table 21. Supported Memory Protocols**

| Protocol              | Hard Controller and Hard PHY | Soft Controller and Hard PHY | PHY Only |
|-----------------------|------------------------------|------------------------------|----------|
| DDR4                  | Yes                          | —                            | Yes      |
| DDR3                  | Yes                          | —                            | Yes      |
| LPDDR3                | Yes                          | —                            | Yes      |
| RDRAM 3               | —                            | Third party                  | Yes      |
| QDR II/II+/II+ Xtreme | —                            | Yes                          | —        |
| QDR-IV                | —                            | Yes                          | —        |

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Empirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.

ISO  
9001:2015  
Registered



Memory protocols not listed above are not supported by the Stratix 10 EMIF IP; however, you can implement a custom memory interface for these protocols using the PHYLite Megafunction.

## 2.2. Stratix 10 EMIF IP Support for 3DS/TSV DDR4 Devices

The Stratix 10 EMIF IP supports high-density DDR4 memory devices with three-dimensional stacked-die (3DS) memory technology in RDIMM format. Both 2H and 4H 3DS devices are supported.

To configure a DDR4 RDIMM memory interface to use 3DS devices, do one of the following:

- Select one of the 3DS-enabled IP presets available in the parameter editor.
- 1. Specify a non-zero value for the **Chip ID Width** parameter on the **Memory** tab in the parameter editor.
- 2. Enter the 3DS-specific specific timing parameters (tRDD\_dir, tFAW\_dir, tRFC\_dir) from the memory vendor's data sheet appropriately.

## 2.3. Migrating to Stratix 10 from Previous Device Families

There is no automatic migration mechanism for external memory interface IP generated for previous device families.

To migrate an existing EMIF IP from an earlier device family to Stratix 10, you must reparameterize and regenerate the IP targeting Stratix 10, using either the IP Catalog or Qsys. If you attempt to recompile an existing IP generated for a previous device family, you will encounter errors in the Quartus Prime software.

UniPHY-based IP continues to be supported for previous device families.

## 2.4. Stratix 10 EMIF Architecture: Introduction

The Stratix 10 EMIF architecture contains many new hardware features designed to meet the high-speed requirements of emerging memory protocols, while consuming the smallest amount of core logic area and power.

The following are key hardware features of the Stratix 10 EMIF architecture:

### Hard Sequencer

The sequencer employs a hard Nios II processor, and can perform memory calibration for a wide range of protocols. You can share the sequencer among multiple memory interfaces of the same or different protocols.

**Note:** You cannot use the hard Nios II processor for any user applications after calibration is complete.

### Hard PHY

The hard PHY in Stratix 10 devices can interface with external memories running at speeds of up to 1.3 GHz. The PHY circuitry is hardened in the silicon, which simplifies the challenges of achieving timing closure and minimal power consumption.



### Hard Memory Controller

The hard memory controller reduces latency and minimizes core logic consumption in the external memory interface. The hard memory controller supports the DDR3, DDR4, and LPDDR3 memory protocols.

### PHY-Only Mode

Protocols that use a hard controller (DDR4, DDR3, LPDDR3, and RLDRAM 3), provide a PHY-only option, which generates only the PHY and sequencer, but not the controller. This PHY-only mode provides a mechanism by which to integrate your own custom soft controller.

### High-Speed PHY Clock Tree

Dedicated high speed PHY clock networks clock the I/O buffers in Stratix 10 EMIF IP. The PHY clock trees exhibit low jitter and low duty cycle distortion, maximizing the data valid window.

### Automatic Clock Phase Alignment

Automatic clock phase alignment circuitry dynamically adjusts the clock phase of core clock networks to match the clock phase of the PHY clock networks. The clock phase alignment circuitry minimizes clock skew that can complicate timing closure in transfers between the FPGA core and the periphery.

### Resource Sharing

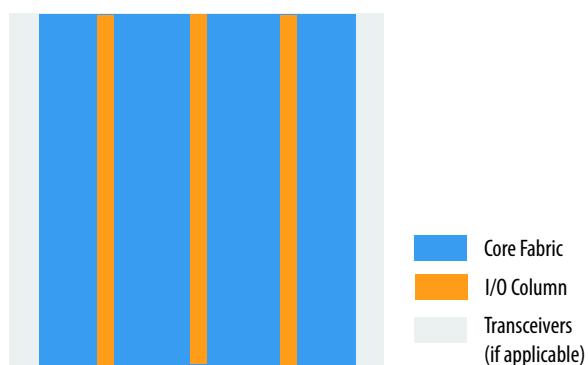
The Stratix 10 architecture simplifies resource sharing between memory interfaces. Resources such as the OCT calibration block, PLL reference clock pin, and core clock can be shared. The hard Nios processor in the I/O subsystem manager (I/O SSM) must be shared across all interfaces in a column.

## 2.4.1. Stratix 10 EMIF Architecture: I/O Subsystem

The I/O subsystem consists of three columns inside the core of Stratix 10 devices.

Each column can be thought of as loosely analogous to an I/O bank.

**Figure 30. Stratix 10 I/O Subsystem**



The I/O subsystem provides the following features:

- General-purpose I/O registers and I/O buffers
- On-chip termination control (OCT)
- I/O PLLs for external memory interfaces and user logic
- Low-voltage differential signaling (LVDS)
- External memory interface components, as follows:
  - Hard memory controller
  - Hard PHY
  - Hard Nios processor and calibration logic
  - DLL

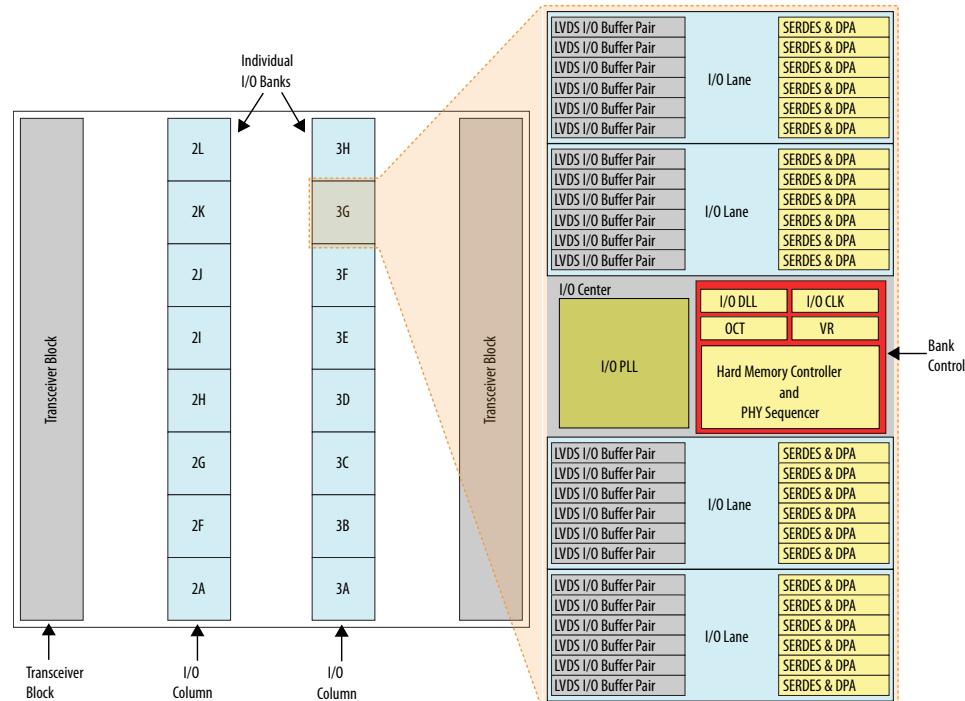
#### 2.4.2. Stratix 10 EMIF Architecture: I/O Column

Stratix 10 devices have two I/O columns, which contain the hardware related to external memory interfaces.

Each I/O column contains the following major parts:

- A hardened Nios processor with dedicated memory. This Nios block is referred to as the I/O SSM.
- Up to 13 I/O banks. Each I/O bank contains the hardware necessary for an external memory interface.

**Figure 31. I/O Column**





### 2.4.3. Stratix 10 EMIF Architecture: I/O SSM

Each column includes one I/O subsystem manager (I/O SSM), which contains a hardened Nios II processor with dedicated memory. The I/O SSM is responsible for calibration of all the EMIFs in the column.

The I/O SSM includes dedicated memory which stores both the calibration algorithm and calibration run-time data. The hardened Nios II processor and the dedicated memory can be used only by an external memory interface, and cannot be employed for any other use. The I/O SSM can interface with soft logic, such as the debug toolkit, via an Avalon-MM bus.

The I/O SSM is clocked by an on-die oscillator, and therefore does not consume a PLL.

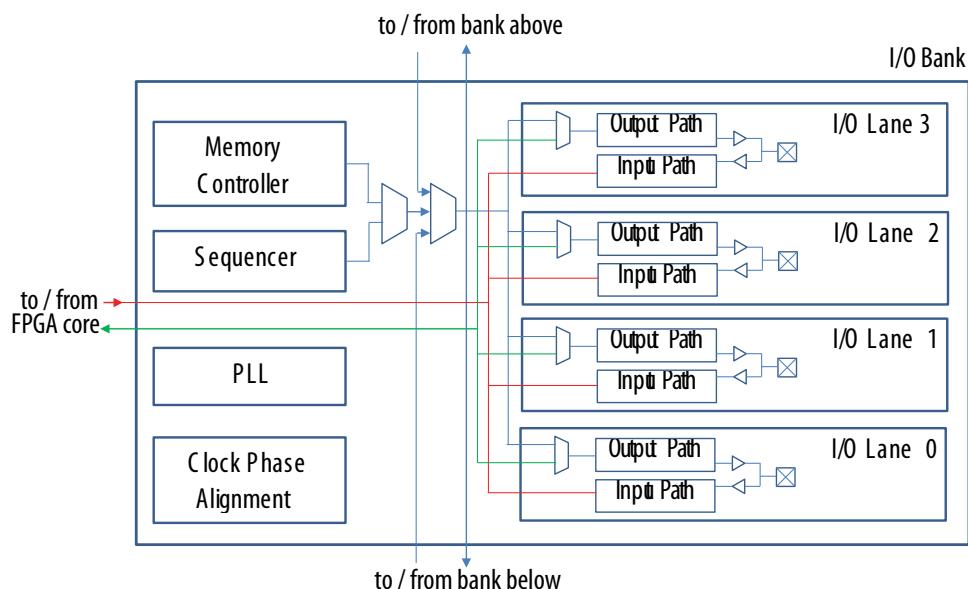
### 2.4.4. Stratix 10 EMIF Architecture: I/O Bank

A single I/O bank contains all the hardware needed to build an external memory interface. Each I/O column contains up to 13 I/O banks; the exact number of banks depends on device size and pin package. You can make a wider interface by connecting multiple banks together.

Each I/O bank resides in an I/O column, and contains the following components:

- Hard memory controller
- Sequencer components
- PLL and PHY clock trees
- DLL
- Input DQS clock trees
- 48 pins, organized into four I/O lanes of 12 pins each

**Figure 32. I/O Bank Architecture in Stratix 10 Devices**





### I/O Bank Usage

The pins in an I/O bank can serve as address and command pins, data pins, or clock and strobe pins for an external memory interface. You can implement a narrow interface, such as a DDR3 or DDR4 x8 interface, with only a single I/O bank. A wider interface, such as x72 or x144, can be implemented by configuring multiple adjacent banks in a multi-bank interface. Any pins in a bank which are not used by the external memory interface remain available for use as general purpose I/O pins (of the same voltage standard).

Every I/O bank includes a hard memory controller which you can configure for DDR3 or DDR4. In a multi-bank interface, only the controller of one bank is active; controllers in the remaining banks are turned off to conserve power.

To use a multi-bank Stratix 10 EMIF interface, you must observe the following rules:

- Designate one bank as the address and command bank.
- The address and command bank must contain all the address and command pins.
- The locations of individual address and command pins within the address and command bank must adhere to the pin map defined in the pin table— regardless of whether you use the hard memory controller or not.
- If you do use the hard memory controller, the address and command bank contains the active hard controller.

All the I/O banks in a column are capable of functioning as the address and command bank. However, for minimal latency, you should select the center-most bank of the interface as the address and command bank.

### 2.4.5. Stratix 10 EMIF Architecture: I/O Lane

An I/O bank contains 48 I/O pins, organized into four I/O lanes of 12 pins each.

Each I/O lane can implement one x8/x9 read capture group (DQS group), with two pins functioning as the read capture clock/strobe pair (DQS/DQS#), and up to 10 pins functioning as data pins (DQ and DM pins). To implement x18 and x36 groups, you can use multiple lanes within the same bank.

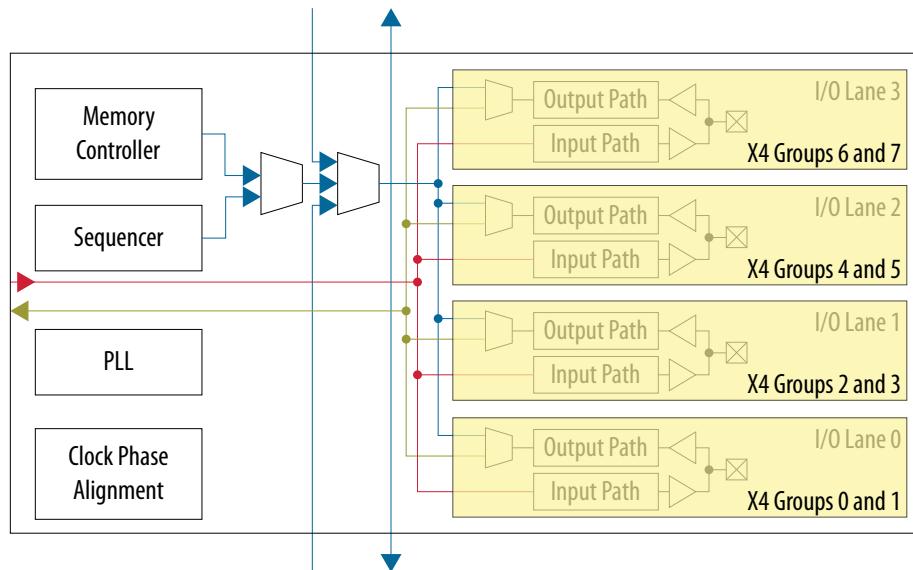
It is also possible to implement a pair of x4 groups in a lane. In this case, four pins function as clock/strobe pair, and 8 pins function as data pins. DM is not available for x4 groups. There must be an even number of x4 groups for each interface.

For x4 groups, DQS0 and DQS1 must be placed in the same I/O lane as a pair. Similarly, DQS2 and DQS3 must be paired. In general, DQS( $x$ ) and DQS( $x+1$ ) must be paired in the same I/O lane.

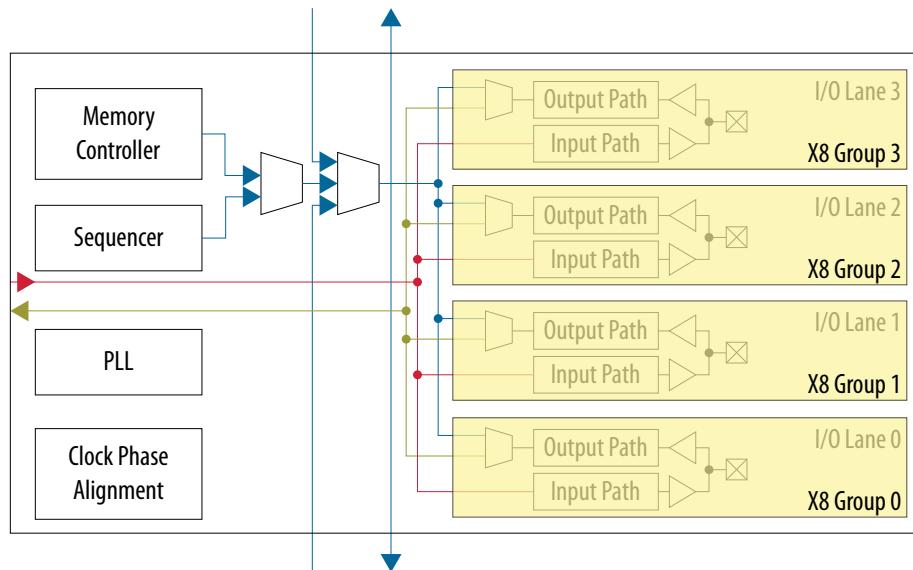
**Table 22. Lanes Used Per Group**

| Group Size | Number of Lanes Used | Maximum Number of Data Pins per Group |
|------------|----------------------|---------------------------------------|
| x8 / x9    | 1                    | 10                                    |
| x18        | 2                    | 22                                    |
| x36        | 4                    | 46                                    |
| pair of x4 | 1                    | 4 per group, 8 per lane               |

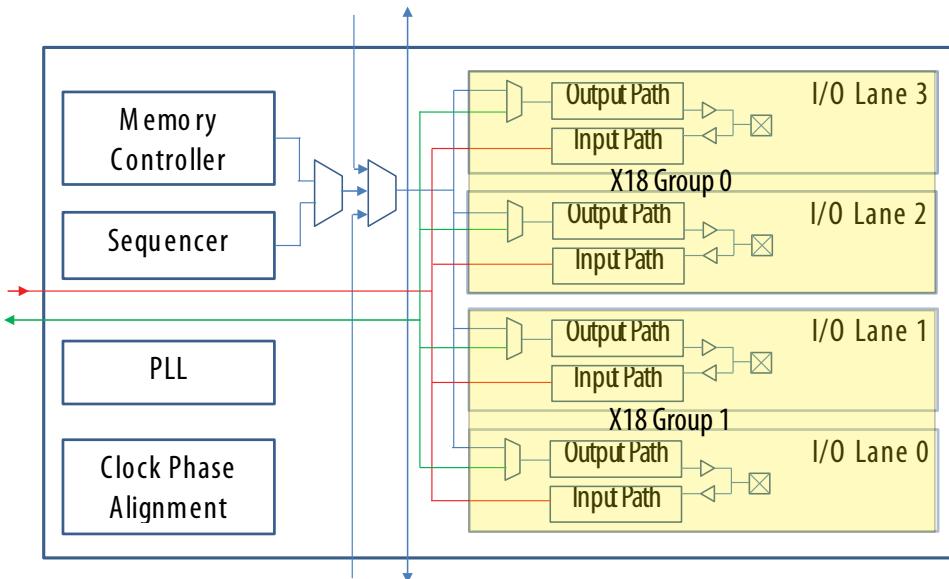
**Figure 33. x4 Group**



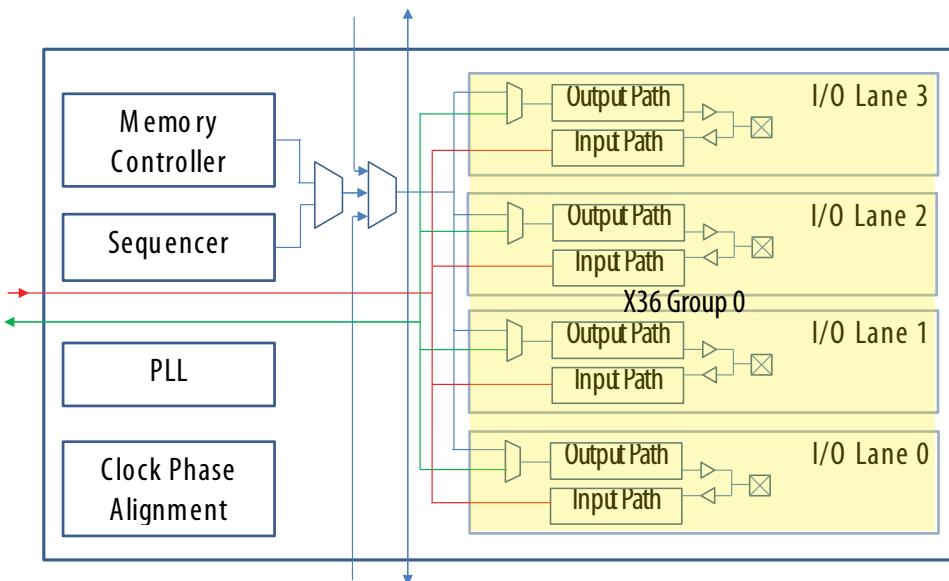
**Figure 34. x8 Group**



**Figure 35. x18 Group**



**Figure 36. x36 Group**



## 2.4.6. Stratix 10 EMIF Architecture: Input DQS Clock Tree

The input DQS clock tree is a balanced clock network that distributes the read capture clock and strobe from the external memory device to the read capture registers inside the I/Os.

You can configure an input DQS clock tree in x4 mode, x8/x9 mode, x18 mode, or x36 mode.



Within every bank, only certain physical pins at specific locations can drive the input DQS clock trees. The pin locations that can drive the input DQS clock trees vary, depending on the size of the group.

**Table 23. Pins Usable as Read Capture Clock / Strobe Pair**

| Group Size | Index of Lanes Spanned by Clock Tree | In-Bank Index of Pins Usable as Read Capture Clock / Strobe Pair |              |
|------------|--------------------------------------|--|--------------|
|            |                                      | Positive Leg   | Negative Leg |
| x4         | 0A                                   | 4  | 5            |
| x4         | 0B                                   | 8  | 9            |
| x4         | 1A                                   | 16   | 17           |
| x4         | 1B                                   | 20   | 21           |
| x4         | 2A                                   | 28   | 29           |
| x4         | 2B                                   | 32   | 33           |
| x4         | 3A                                   | 40   | 41           |
| x4         | 3B                                   | 44   | 45           |
| x8 / x9    | 0                                    | 4  | 5            |
| x8 / x9    | 1                                    | 16   | 17           |
| x8 / x9    | 2                                    | 28   | 29           |
| x8 / x9    | 3                                    | 40   | 41           |
| x18        | 0, 1                                 | 12   | 13           |
| x18        | 2, 3                                 | 36   | 37           |
| x36        | 0, 1, 2, 3                           | 20   | 21           |

#### 2.4.7. Stratix 10 EMIF Architecture: PHY Clock Tree

Dedicated high-speed clock networks drive I/Os in Stratix 10 EMIF. Each PHY clock network spans only one bank.

The relatively short span of the PHY clock trees results in low jitter and low duty-cycle distortion, maximizing the data valid window.

The PHY clock tree in Stratix 10 devices can run as fast as 1.3 GHz. All Stratix 10 external memory interfaces use the PHY clock trees.

#### 2.4.8. Stratix 10 EMIF Architecture: PLL Reference Clock Networks

Each I/O bank includes a PLL that can drive the PHY clock trees of that bank, through dedicated connections. In addition to supporting EMIF-specific functions, such PLLs can also serve as general-purpose PLLs for user logic.

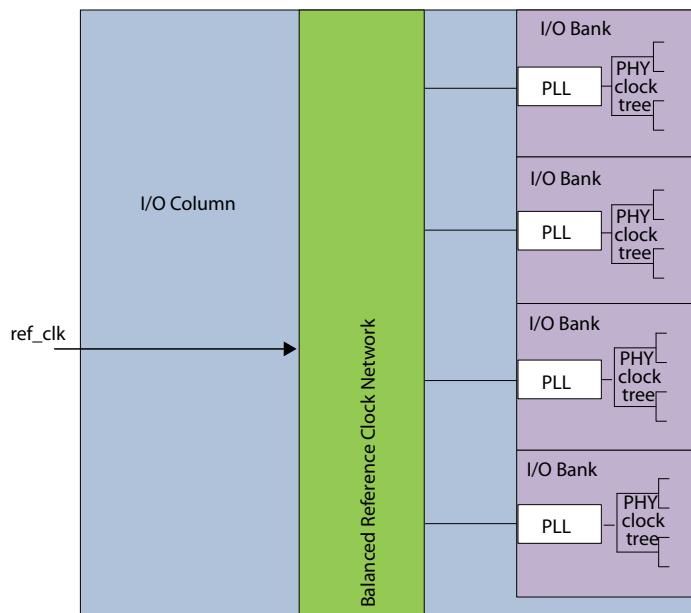
Stratix 10 external memory interfaces that span multiple banks use the PLL in each bank. (Some previous device families relied on a single PLL with clock signals broadcast to all I/Os by a clock network.) The Stratix 10 architecture allows for relatively short PHY clock networks, reducing jitter and duty-cycle distortion.

The following mechanisms ensure that the clock outputs of individual PLLs in a multi-bank interface remain in phase:

- A single PLL reference clock source feeds all PLLs. The reference clock signal reaches the PLLs by a balanced PLL reference clock tree. The Quartus Prime software automatically configures the PLL reference clock tree so that it spans the correct number of banks.
- The EMIF IP sets the PLL M and N values appropriately to maintain synchronization among the clock dividers across the PLLs. This requirement restricts the legal PLL reference clock frequencies for a given memory interface frequency and clock rate. The Stratix 10 EMIF IP parameter editor automatically calculates and displays the set of legal PLL reference clock frequencies. If you plan to use an on-board oscillator, you must ensure that its frequency matches the PLL reference clock frequency that you select from the displayed list. The correct M and N values of the PLLs are set automatically based on the PLL reference clock frequency that you select.

**Note:** The PLL reference clock pin may be placed in the address and command I/O bank or in a data I/O bank; there is no implication on timing. However, for debug flexibility, it is recommended to place the PLL reference clock in the address and command I/O bank.

**Figure 37. PLL Balanced Reference Clock Tree**



#### 2.4.9. Stratix 10 EMIF Architecture: Clock Phase Alignment

In Stratix 10 external memory interfaces, a global clock network clocks registers inside the FPGA core, and the PHY clock network clocks registers inside the FPGA periphery. Clock phase alignment circuitry employs negative feedback to dynamically adjust the phase of the core clock signal to match the phase of the PHY clock signal.

The clock phase alignment feature effectively eliminates the clock skew effect in all transfers between the core and the periphery, facilitating timing closure. All Stratix 10 external memory interfaces employ clock phase alignment circuitry.



Figure 38. Clock Phase Alignment Illustration

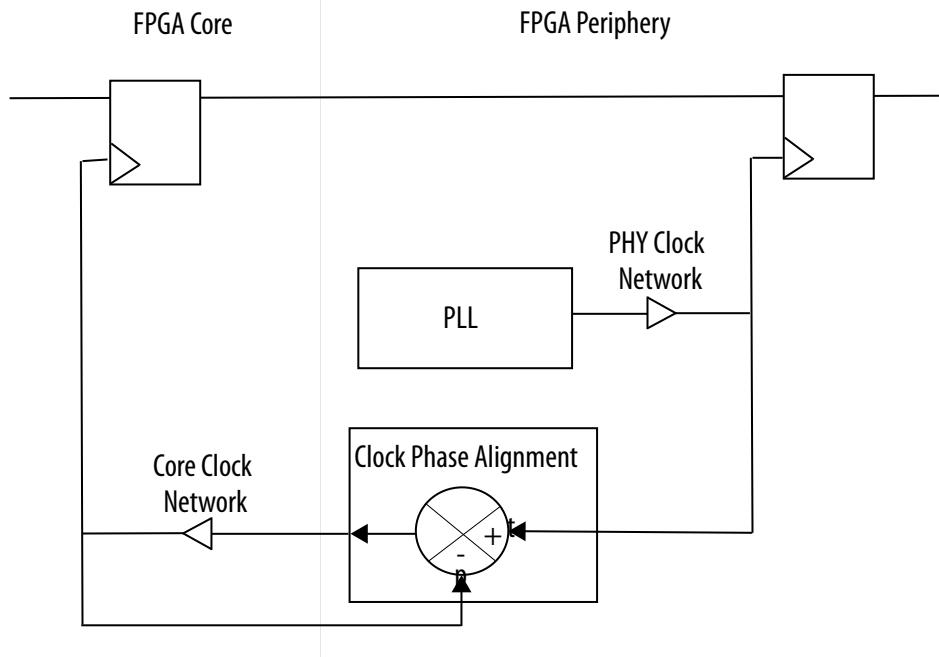
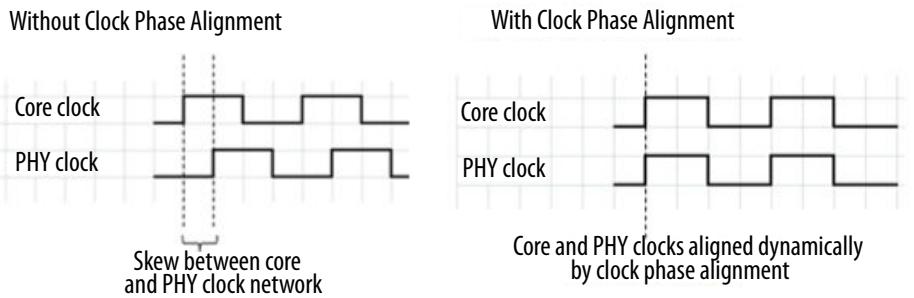


Figure 39. Effect of Clock Phase Alignment



## 2.5. Hardware Resource Sharing Among Multiple Stratix 10 EMIFs

Often, it is necessary or desirable to share certain hardware resources between interfaces.

### 2.5.1. I/O SSM Sharing

The I/O SSM contains a hard Nios-II processor and dedicated memory storing the calibration software code and data.

When a column contains multiple memory interfaces, the hard Nios-II processor calibrates each interface serially. Interfaces placed within the same I/O column always share the same I/O SSM. The Quartus Prime Fitter handles I/O SSM sharing automatically.

## 2.5.2. I/O Bank Sharing

Data lanes from multiple compatible interfaces can share a physical I/O bank to achieve a more compact pin placement. To share an I/O bank, interfaces must use the same memory protocol, rate, frequency, I/O standard, and PLL reference clock signal.

### Rules for Sharing I/O Banks

- A bank cannot serve as the address and command bank for more than one interface. This means that lanes which implement address and command pins for different interfaces cannot be allocated to the same physical bank.

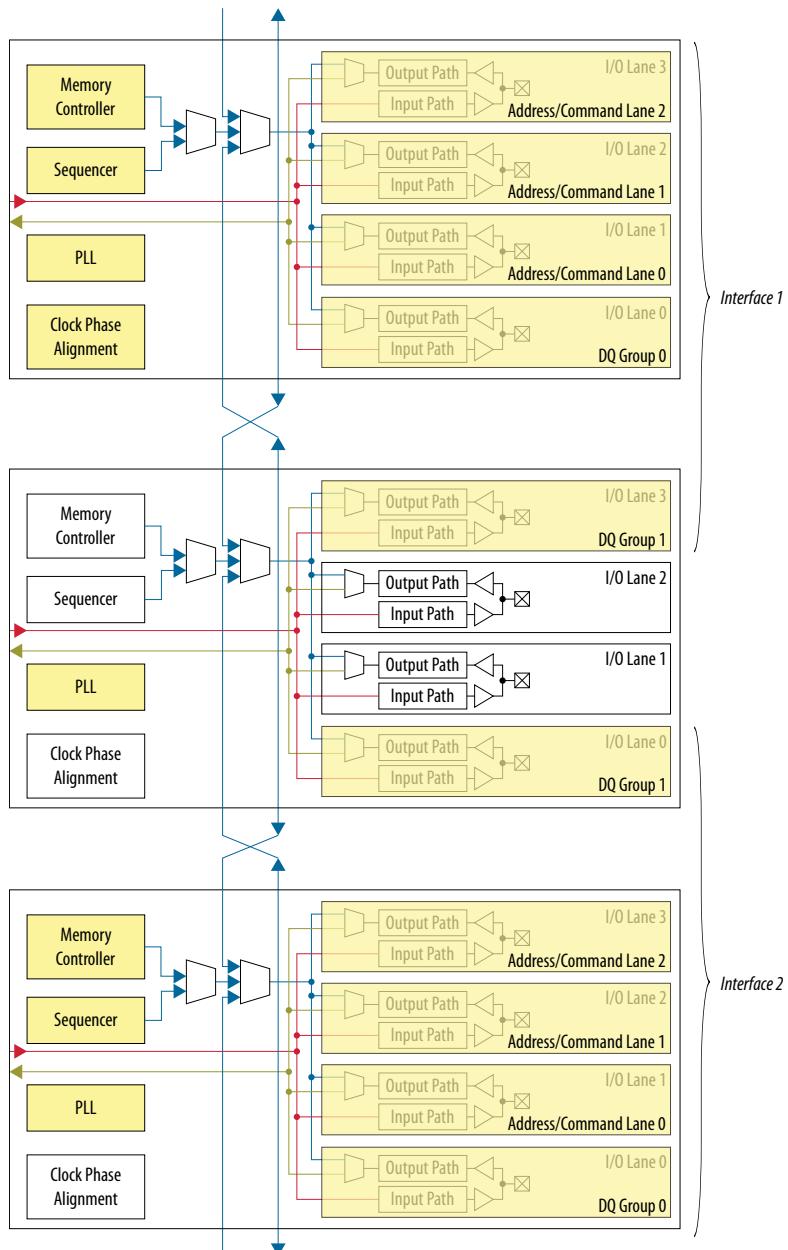
*Note:* An exception to the above rule exists when two interfaces are configured in a Ping-Pong PHY fashion. In such a configuration, two interfaces share the same set of address and command pins, effectively meaning that they share the same address and command tile.

- Pins within a lane cannot be shared by multiple memory interfaces.
- Pins that are not used by EMIF IP can serve as general-purpose I/Os of compatible voltage and termination settings.
- You can configure a bank as LVDS or as EMIF, but not both at the same time.
- Interfaces that share banks must reside at consecutive bank locations.

The following diagram illustrates two x16 interfaces sharing an I/O bank. The two interfaces share the same clock phase alignment block, so that one core clock signal can interact with both interfaces. Without sharing, the two interfaces would occupy a total of four physical banks instead of three.



**Figure 40. I/O Bank Sharing**



### 2.5.3. PLL Reference Clock Sharing

In Stratix 10, every I/O bank contains a PLL, meaning that it is not necessary to share PLLs in the interest of conserving resources. Nonetheless, it is often desirable to share PLLs for other reasons.



You might want to share PLLs between interfaces for the following reasons:

- To conserve pins.
- When combined with the use of the balanced PLL reference clock tree, to allow the clock signals at different interfaces to be synchronous and aligned to each other. For this reason, interfaces that share core clock signals must also share the PLL reference clock signal.

To implement PLL reference clock sharing, in your RTL code connect the PLL reference clock signal at your design's top-level to the PLL reference clock port of multiple interfaces.

To share a PLL reference clock, the following requirements must be met:

- Interfaces must expect a reference clock signal of the same frequency.
- Interfaces must be placed in the same column.
- Interfaces must be placed at adjacent bank locations.

#### 2.5.4. Core Clock Network Sharing

It is often desirable or necessary for multiple memory interfaces to be accessible using a single clock domain in the FPGA core.

You might want to share core clock networks for the following reasons:

- To minimize the area and latency penalty associated with clock domain crossing.
- To minimize consumption of core clock networks.

Multiple memory interfaces can share the same core clock signals under the following conditions:

- The memory interfaces have the same protocol, rate, frequency, and PLL reference clock source.
- The interfaces reside in the same I/O column.
- The interfaces reside in adjacent bank locations.

For multiple memory interfaces to share core clocks, you must specify one of the interfaces as master and the remaining interfaces as slaves. Use the `Core_clocks_sharing` setting in the parameter editor to specify the master and slaves.

In your RTL, connect the `clks_sharing_master_out` signal from the master interface to the `clks_sharing_slave_in` signal of all the slave interfaces. Both the master and slave interfaces expose their own output clock ports in the RTL (e.g. `emif_usr_clk`, `afi_clk`), but the signals are equivalent, so it does not matter whether a clock port from a master or a slave is used.

Core clock sharing necessitates PLL reference clock sharing; therefore, only the master interface exposes an input port for the PLL reference clock. All slave interfaces use the same PLL reference clock signal.



## 2.6. Stratix 10 EMIF IP Component

The external memory interface IP component for Stratix 10 provides a complete solution for implementing external memory interfaces. The EMIF IP also includes a protocol-specific calibration algorithm that automatically determines the optimal delay settings for a robust external memory interface.

The external memory interface IP comprises the following parts:

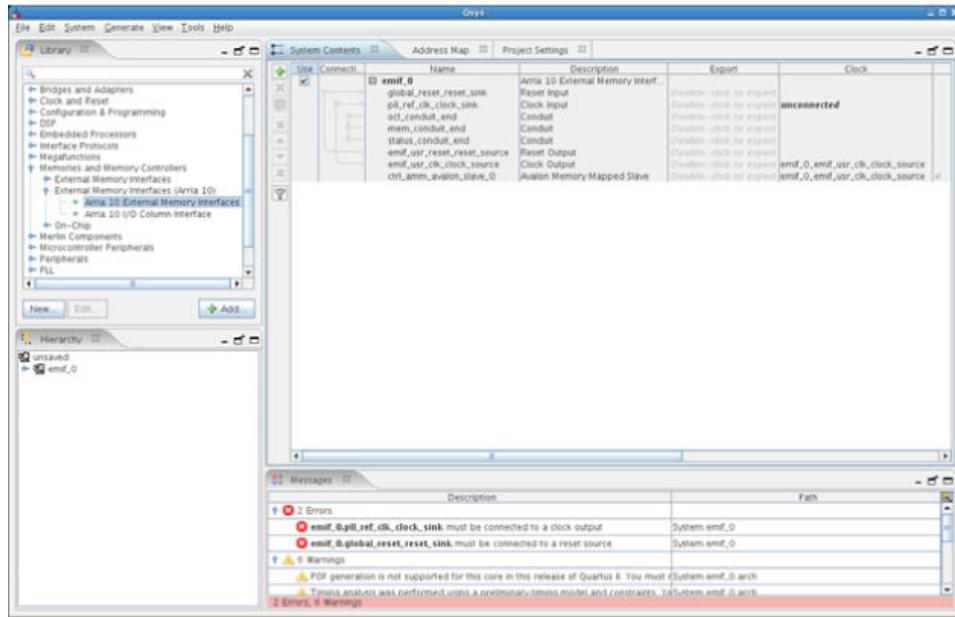
- A set of synthesizable files that you can integrate into a larger design
- A stand-alone synthesizable example design that you can use for hardware validation
- A set of simulation files that you can incorporate into a larger project
- A stand-alone simulation example project that you can use to observe controller and PHY operation
- A set of timing scripts that you can use to determine the maximum operating frequency of the memory interface based on external factors such as board skew, trace delays, and memory component timing parameters
- A customized data sheet specific to your memory interface configuration

### 2.6.1. Instantiating Your Stratix 10 EMIF IP in a Qsys Project

The following steps describe how to instantiate your Stratix 10 EMIF IP in a Qsys project.

1. Within the Qsys interface, select **Memories and Memory Controllers** in the component **Library** tree.
2. Under **Memories and Memory Controllers**, select **External Memory Interfaces (Stratix 10)**.
3. Under **External Memory Interfaces (Stratix 10)**, select the **Stratix 10 External Memory Interface** component.

**Figure 41.** Instantiating Stratix 10 EMIF IP in Qsys



### 2.6.1.1. Logical Connections

The following logical connections exist in a Stratix 10 EMIF IP core.

**Table 24. Logical Connections Table**

| Logical Connection                              | Description  |
|---|--|
| afi_conduit_end<br>(Conduit)                    | The Altera PHY Interface (AFI) connects a memory controller to the PHY. This interface is exposed only when you configure the memory interface in PHY-Only mode. The interface is synchronous to the afi_clk clock and afi_reset_n reset.  |
| afi_clk_conduit_end<br>(Conduit)                | Use this clock signal to clock the soft controller logic. The afi_clk is an output clock coming from the PHY when the memory interface is in PHY-Only mode. The phase of afi_clk is adjusted dynamically by hard circuitry for the best data transfer between FPGA core logic and periphery logic with maximum timing margin. Multiple memory interface instances can share a single afi_clk using the <b>Core Clocks Sharing</b> option during IP generation. |
| afi_half_clk_conduit_end<br>(Conduit)           | This clock runs at half the frequency of afi_1k. It is exposed only when the memory interface is in PHY-Only mode.   |
| afi_reset_n_conduit_end<br>(Conduit)            | This single-bit reset provides a synchronized reset output. Use this signal to reset all registers that are clocked by either afi_clk or afi_half_clk.   |
| cal_debug_avalon_slave<br>(Avalon Slave/Target) | This interface is exposed when the <b>EMIF Debug Toolkit/On-chip Debug Port</b> option is set to <b>Export</b> . This interface can be connected to a Stratix 10 external memory interface debug component to allow EMIF Debug Toolkit access, or it can be used directly by user logic to access calibration diagnostic data.   |
| cal_debug_clk_clock_sink<br>(Clock Input)       | This clock is used for the cal_debug_avalon_slave interface. It can be connected to the emif_usr_clk_clock_source interface.   |
| cal_debug_reset_reset_sink<br>(Reset Input)     | This reset is used for the cal_debug_avalon_slave interface. It can be connected to the emif_usr_reset_reset_source interface.   |

*continued...*



| Logical Connection                                    | Description   |
|---|---|
| cal_debug_out_avalon_master<br>(Avalon Master/Source) | This interface is exposed when the <b>Enable Daisy-Chaining for EMIF Debug Toolkit/On-Chip Debug Port</b> option is enabled. Connect this interface to the cal_debug_avalon_slave interface of the next EMIF instance in the same I/O column.   |
| cal_debug_out_clk_clock_source<br>(Clock Output)      | This interface should be connected to the cal_debug_clk_clock_sink interface of the next EMIF instance in the same I/O column (similar to cal_debug_out_avalon_master).   |
| cal_debug_out_reset_reset_source<br>(Reset Output)    | This interface should be connected to the cal_debug_reset_reset_sink interface of the next EMIF instance in the same I/O column (similar to cal_debug_out_avalon_master).   |
| effmon_csr_avalon_slave<br>(Avalon Slave/Target)      | This interface allows access to the Efficiency Monitor CSR. For more information, see the documentation on the UniPHY Efficiency Monitor.   |
| global_reset_reset_sink<br>(Reset Input)              | This single-wire input port is the asynchronous reset input for the EMIF core.  |
| pll_ref_clk_clock_sink<br>(Clock Input)               | This single-wire input port connects the external PLL reference clock to the EMIF core. Multiple EMIF cores may share a PLL reference clock source, provided the restrictions outlined in the <i>PLL Reference Clock Network</i> section are observed.  |
| oct_conduit_end<br>(Conduit)                          | This logical port is connected to an OCT pin and provides calibrated reference data for EMIF cores with pins that use signaling standards that require on-chip termination. Depending on the I/O standard, reference voltage, and memory protocol, multiple EMIF cores may share a single OCT pin.  |
| mem_conduit_end<br>(Conduit)                          | This logical conduit can attach an Intel memory model to an EMIF core for simulation. Memory models for various protocols are available under the <b>Memories and Memory Controllers — External Memory Interfaces — Memory Models</b> section of the component library in Qsys. You must ensure that all configuration parameters for the memory model match the configuration parameters of the EMIF core.   |
| status_conduit_end<br>(Conduit)                       | The status conduit exports two signals that you can sample to determine whether the calibration operation passed or failed for that core.   |
| emif_usr_reset_reset_source<br>(Reset Output)         | You should use this single-bit, synchronized, reset output to reset all components that are synchronously connected to the EMIF core. Assertion of the global reset input triggers an assertion of this output as well, therefore you should rely on this signal only as a reset source for all components connected to the EMIF core.  |
| emif_usr_clk_clock_source<br>(Clock Output)           | Use this single-bit clock output to clock all logic connected to the EMIF core. The phase of this clock signal is adjusted dynamically by circuitry in the EMIF core such that data can be transferred between core logic and periphery registers with maximum timing margin. Drive all logic connected to the EMIF core with this clock signal. Other clock sources generated from the same reference clock or even the same PLL may have unknown phase relationships. Multiple EMIF cores can share a single core clock using the Core Clocks Sharing option described in the Example Design tab of the parameter editor.   |
| ctrl_amm_avalon_slave<br>(Avalon Slave/Target)        | This Avalon target port initiates read or write commands to the controller. Refer to the Avalon Interface Specification for more information on how to design cores that comply to the Avalon Bus Specification.<br><br>For DDR3, DDR4, and LPDDR3 protocols with the hard PHY and hard controller configuration and an AVL slave interface exposed, ctrl_amm_avalon_slave is renamed to ctrl_amm_avalon_slave_0.<br><br>For QDR II, QDR II+, and QDR II+ Xtreme interfaces with hard PHY and soft controller, separate read and write connections are used.<br>ctrl_amm_avalon_slave_0 is the read port and ctrl_amm_avalon_slave_1 is the write port.<br><br>For QDR-IV interfaces with hard PHY and soft controller operating at quarter rate, a total of eight separate Avalon interfaces (named ctrl_amm_avalon_slave_0 to ctrl_amm_avalon_slave_7) are used to maximize bus efficiency. |



### Related Information

[Avalon Interface Specifications](#)

## 2.6.2. File Sets

The Stratix 10 EMIF IP core generates four output file sets for every EMIF IP core, arranged according to the following directory structure.

**Table 25. Generated File Sets**

| Directory                            | Description   |
|--------------------------------------|---|
| <core_name>/*                        | This directory contains only the files required to integrate a generated EMIF core into a larger design. This directory contains: <ul style="list-style-type: none"><li>• Synthesizable HDL source files</li><li>• Customized TCL timing scripts specific to the core (protocol and topology)</li><li>• HEX files used by the calibration algorithm to identify the interface parameters</li><li>• A customized data sheet that describes the operation of the generated core</li></ul> <p>Note: The top-level HDL file is generated in the root folder as &lt;core_name&gt;.v (or &lt;core_name&gt;.vhdl for VHDL designs). You can reopen this file in the parameter editor if you want to modify the EMIF core parameters and regenerate the design.</p> |
| <core_name>_sim/*                    | This directory contains the simulation fileset for the generated EMIF core. These files can be integrated into a larger simulation project. For convenience, simulation scripts for compiling the core are provided in the /mentor, /cadence, /synopsys, and /riviera subdirectories.<br>The top-level HDL file, <core_name>.v (or <core_name>.vhdl) is located in this folder, and all remaining HDL files are placed in the /altera_emif_arch_nf subfolder, with the customized data sheet. The contents of this directory are not intended for synthesis.  |
| emif_<instance_num>_example_design/* | This directory contains a set of TCL scripts, QSYS project files and README files for the complete synthesis and simulation example designs. You can invoke these scripts to generate a standalone fully-synthesizable project complete with an example driver, or a standalone simulation design complete with an example driver and a memory model.   |

## 2.6.3. Customized readme.txt File

When you generate your Stratix 10 EMIF IP, the system produces a customized readme file, containing data indicative of the settings in your IP core.

The readme file is <variation\_name>/altera\_emif\_arch\_nf\_<version\_number>/<synth/sim>/<variation\_name>\_altera\_emif\_arch\_nf\_<version\_number>\_<unique ID>\_readme.txt, and contains a summary of Stratix 10 EMIF information, and details specific to your IP core, including:

- Pin location guidelines
- External port names, directions, and widths
- Internal port names, directions, and widths



- Avalon interface configuration details (if applicable)
- Calibration mode
- A brief summary of all configuration settings for the generated IP

You should review the generated readme file for implementation guidelines specific to your IP core.

#### 2.6.4. Clock Domains

The Stratix 10 EMIF IP core provides a single clock domain to drive all logic connected to the EMIF core.

The frequency of the clock depends on the core-clock to memory-clock interface rate ratio. For example, a quarter-rate interface with an 800 MHz memory clock would provide a 200 MHz clock to the core ( $800 \text{ MHz} / 4 = 200 \text{ MHz}$ ). The EMIF IP dynamically adjusts the phase of the core clock with respect to the periphery clock to maintain the optimum alignment for transferring data between the core and periphery. Independent EMIF IP cores driven from the same reference clock have independent core clock domains. You should employ one of the following strategies if you are implementing multiple EMIF cores:

1. Treat all crossing between independent EMIF-generated clock domains as asynchronous, even though they are generated from the same reference clock.
2. Use the **Core clock sharing** option to enforce that multiple EMIF cores share the same core clock. You must enable this option before IP generation. This option is applicable only for cores that reside in the same I/O column.

#### 2.6.5. ECC in Stratix 10 EMIF IP

The ECC (error correction code) is a soft component of the Stratix 10 EMIF IP that reduces the chance of errors when reading and writing to external memory. ECC allows correction of single-bit errors and reduces the chances of system failure.

The ECC component includes an encoder, decoder, write FIFO buffer, and modification logic, to allow read-modify-write operations. The ECC code employs standard Hamming logic to correct single-bit errors and to detect double-bit errors. ECC is available in 16, 24, 40, and 72 bit widths.

When writing data to memory, the encoder creates ECC bits and writes them together with the regular data. When reading from memory, the decoder checks the ECC bits and regular data, and passes the regular data unchanged if no errors are detected. If a single-bit error is detected, the ECC logic corrects the error and passes the regular data. If more than a single-bit error is detected, the ECC logic sets a flag to indicate the error.

Read-modify-write operations can occur in the following circumstances:

- A partial write in data mask mode (with or without ECC), where at least one memory burst of byte-enable is not all ones or all zeros.
- Auto-correction with ECC enabled. This is usually a dummy write issued by the auto-correction logic when a single-bit error is detected.

Read-modify-write operations can have a significant effect on memory interface performance; for best performance, you should minimize partial writes.



### 2.6.5.1. ECC Components

The ECC logic communicates with user logic via an Avalon-MM interface, and with the hard memory controller via an Avalon-ST interface

#### ECC Encoder

The ECC encoder consists of a x64/x72 encoder IP core capable of single-bit error correction and double-bit error detection. The encoder takes 64 bits input and converts it to 72 bits output, where the 8 additional bits are ECC code. The encoder supports any input data width less than 64-bits. Any unused input data bits are set to zero.

#### ECC Decoder

The ECC decoder consists of a x72/x64 decoder IP core capable of double-bit error detection. The decoder takes 72 bits input and converts it to 64 bits output. The decoder also produces single-bit error and double-bit error information. The decoder controls the user read data valid signal; when read data is intended for partial write, the user read data valid signal is deasserted, because the read data is meant for merging, not for the user.

#### Partial Write Data FIFO Buffer

The Partial Write Data FIFO Buffer is implemented in soft logic to store partial write data and byte enable. Data and byte enable are popped and merged with the returned read data. A partial write can occur in the following situations:

- At least one memory burst of byte enable is not all ones or all zeroes.
- Non data masked mode, where all memory bursts of byte enable are not all ones.
- A dummy write with auto-correction logic, where all memory bursts of byte enable are all zeroes. (You might use a dummy write when correcting memory content with a single-bit error.)

#### Merging Logic

Merge return partial read data with write data based on byte enabled popped from the FIFO buffer, and send it to the ECC encoder.

#### Pointer FIFO Buffer

The pointer FIFO buffer is implemented in soft logic to store write data pointers. The ECC logic refers to the pointers when sending write data to the data buffer control (DBC). The pointers serve to overwrite existing write data in the data buffer during a read-modify-write process.

#### Partial Logic

Partial logic decodes byte enable information and distinguishes between normal and partial writes.

#### Memory Mode Register Interface

The Memory Mode Register (MMR) interface is an Avalon-based interface through which core logic can access debug signals and sideband operation requests in the hard memory controller.



The MMR logic routes ECC-related operations to an MMR register implemented in soft logic, and returns the ECC information via an Avalon-MM interface. The MMR logic tracks single-bit and double-bit error status, and provides the following information:

- Interrupt status.
- Single-bit error and double-bit error status.
- Single-bit error and double-bit error counts, to a maximum of 15. (If more than 15 errors occur, the count will overflow.)
- Address of the last error.

### 2.6.5.2. ECC User Interface Controls

You can enable the ECC logic from the **Configuration, Status, and Error Handling** section of the **Controller** tab in the parameter editor.

There are three user interface settings related to ECC:

- **Enabled Memory-Mapped Configuration and Status Register (MMR) Interface:** Allows run-time configuration of the memory controller. You can enable this option together with ECC to retrieve error detection information from the ECC logic.
- **Enabled Error Detection and Correction Logic:** Enables ECC logic for single-bit error correction and double-bit error detection.
- **Enable Auto Error Correction:** Allows the controller to automatically correct single-bit errors detected by the ECC logic.

## 2.7. Examples of External Memory Interface Implementations for DDR4

The following figures are examples of external memory interface implementations for different DDR4 memory widths. The figures show the locations of the address/command and data pins in relation to the locations of the memory controllers.

**Figure 42. DDR4 1x8 Implementation Example (One I/O Bank)**

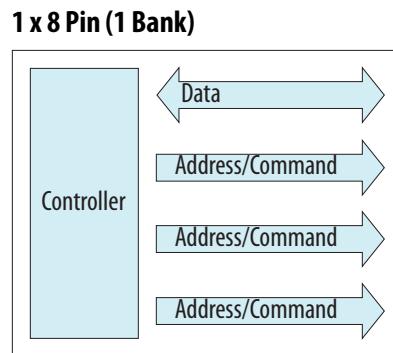


Figure 43. DDR4 1x32 Implementation Example (Two I/O Banks)

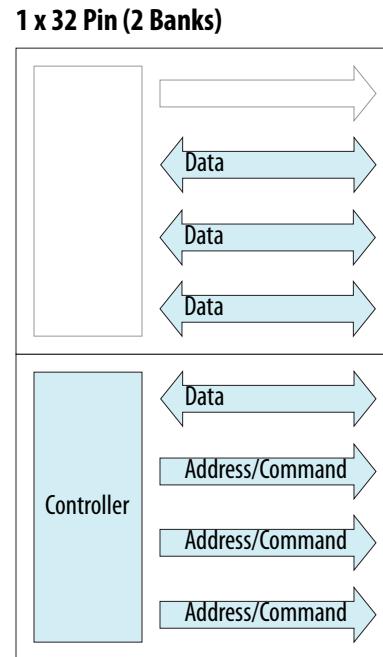
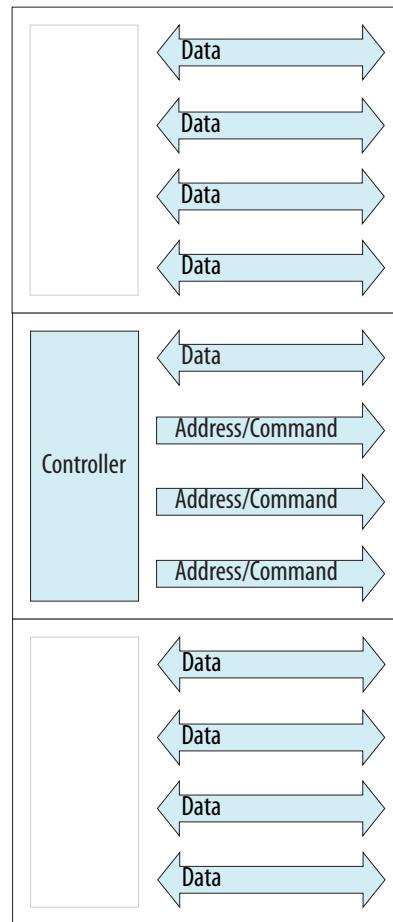


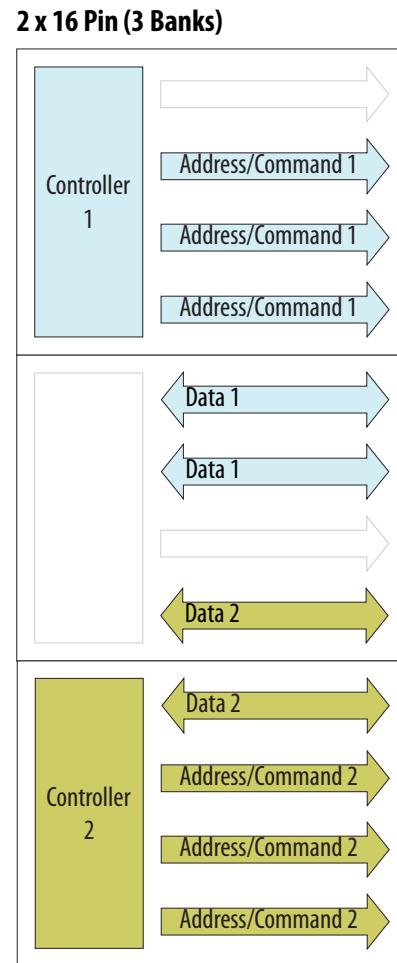


Figure 44. DDR4 1x72 Implementation Example (Three I/O Banks)

1 x 72 Pin (3 Banks)



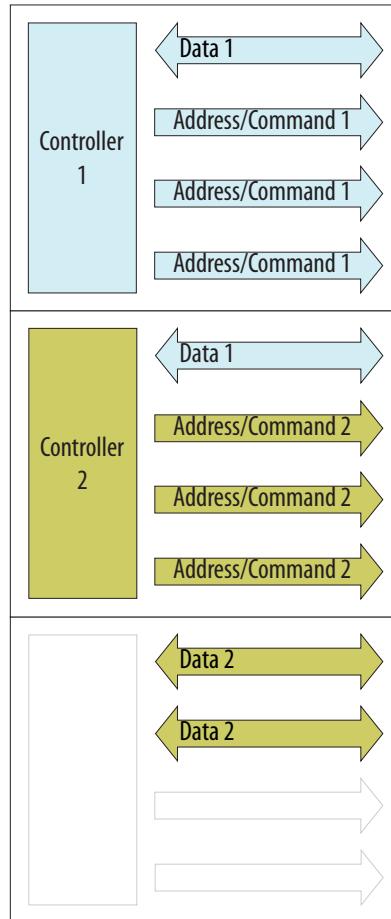
**Figure 45. DDR4 2x16 Implementation Example with Controllers in Non-Adjacent Banks (Three I/O Banks)**

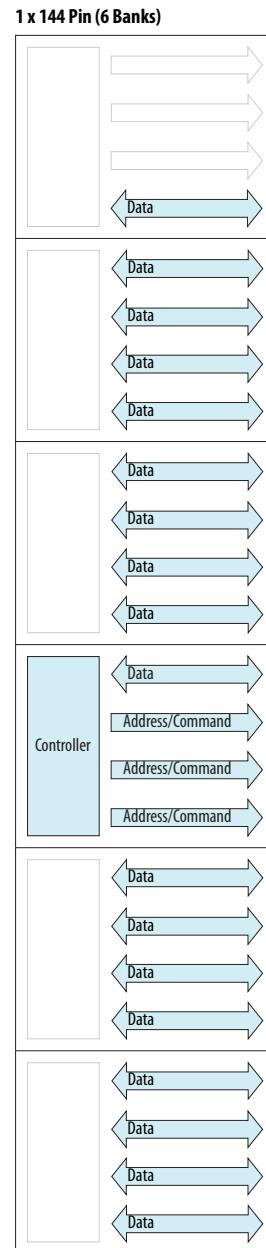




**Figure 46. DDR4 2x16 Implementation Example with Controllers in Adjacent Banks (Three I/O Banks)**

**2 x 16 Pin (3 Banks)**



**Figure 47. DDR4 1x144 Implementation Example (Six I/O Banks)**

## 2.8. Stratix 10 EMIF Sequencer

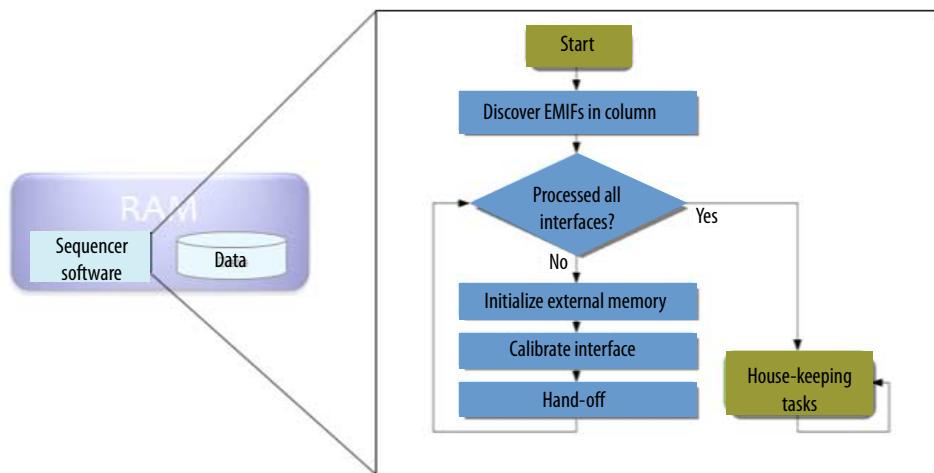
The Stratix 10 EMIF sequencer is fully hardened in silicon, with executable code to handle protocols and topologies. Hardened RAM contains the calibration algorithm.



The Stratix 10 EMIF sequencer is responsible for the following operations:

- Initializes memory devices.
- Calibrates the external memory interface.
- Governs the hand-off of control to the memory controller.
- Handles recalibration requests and debug requests.
- Handles all supported protocols and configurations.

**Figure 48. Stratix 10 EMIF Sequencer Operation**



### 2.8.1. Stratix 10 EMIF DQS Tracking

DQS tracking is enabled for QDR II / II+ / QDR II+ Xtreme, RLDRAM 3, and LPDDR3 protocols. DQS tracking is not available for DDR3 and DDR4 protocols.

## 2.9. Stratix 10 EMIF Calibration

The calibration process compensates for skews and delays in the external memory interface.

The following effects can be compensated for by the calibration process:

- Timing and electrical constraints, such as setup/hold time and  $V_{ref}$  variations.
- Circuit board and package factors, such as skew, fly-by effects, and manufacturing variations.
- Environmental uncertainties, such as variations in voltage and temperature.
- The demanding effects of small margins associated with high-speed operation.

**Note:** The calibration process is intended to maximize margins for robust EMIF operation; it cannot compensate for an inadequate PCB layout.



## 2.9.1. Stratix 10 Calibration Stages

At a high level, the calibration routine consists of address and command calibration, read calibration, and write calibration.

The stages of calibration vary, depending on the protocol of the external memory interface.

**Table 26. Calibration Stages by Protocol**

| Stage                      | DDR4 | DDR3 | LPDDR3 | RLDRAM 3 | QDR-IV | QDR II/II+ |
|----------------------------|------|------|--------|----------|--------|------------|
| <b>Address and command</b> |      |      |        |          |        |            |
| Leveling                   | Yes  | Yes  | —      | —        | —      | —          |
| Deskew                     | Yes  | —    | Yes    | —        | Yes    | —          |
| <b>Read</b>                |      |      |        |          |        |            |
| DQSen                      | Yes  | Yes  | Yes    | Yes      | Yes    | Yes        |
| Deskew                     | Yes  | Yes  | Yes    | Yes      | Yes    | Yes        |
| VREF-In                    | Yes  | —    | —      | —        | Yes    | —          |
| LFIFO                      | Yes  | Yes  | Yes    | Yes      | Yes    | Yes        |
| <b>Write</b>               |      |      |        |          |        |            |
| Leveling                   | Yes  | Yes  | Yes    | Yes      | Yes    | —          |
| Deskew                     | Yes  | Yes  | Yes    | Yes      | Yes    | Yes        |
| VREF-Out                   | Yes  | —    | —      | —        | —      | —          |

## 2.9.2. Stratix 10 Calibration Stages Descriptions

The various stages of calibration perform address and command calibration, read calibration, and write calibration.

### Address and Command Calibration

The goal of address and command calibration is to delay address and command signals as necessary to optimize the address and command window. This stage is not available for all protocols, and cannot compensate for an inefficient board design.

Address and command calibration consists of the following parts:

- Leveling calibration— Centers the CS# signal and the entire address and command bus, relative to the CK clock. This operation is available for DDR3 and DDR4 interfaces only.
- Deskew calibration— Provides per-bit deskew for the address and command bus (except CS#), relative to the CK clock. This operation is available for DDR4 and QDR-IV interfaces only.



### Read Calibration

Read calibration consists of the following parts:

- DQSen calibration— Calibrates the timing of the read capture clock gating and ungating, so that the PHY can gate and ungate the read clock at precisely the correct time—if too early or too late, data corruption can occur. The algorithm for this stage varies, depending on the memory protocol.
- Deskew calibration— Performs per-bit deskew of read data relative to the read strobe or clock.
- VREF-In calibration— Calibrates the VREF level at the FPGA.
- LFIFO calibration: Normalizes differences in read delays between groups due to fly-by, skews, and other variables and uncertainties.

### Write Calibration

Write calibration consists of the following parts:

- Leveling calibration— Aligns the write strobe and clock to the memory clock, to compensate for skews, especially those associated with fly-by topology. The algorithm for this stage varies, depending on the memory protocol.
- Deskew calibration— Performs per-bit deskew of write data relative to the write strobe and clock.
- VREF-Out calibration— Calibrates the VREF level at the memory device.

### 2.9.3. Stratix 10 Calibration Algorithms

The calibration algorithms sometimes vary, depending on the targeted memory protocol.

#### Address and Command Calibration

Address and command calibration consists of the following parts:



- Leveling calibration— (DDR3 and DDR4 only) Toggles the CS# and CAS# signals to send read commands while keeping other address and command signals constant. The algorithm monitors for incoming DQS signals, and if the DQS signal toggles, it indicates that the read commands have been accepted. The algorithm then repeats using different delay values, to find the optimal window.
- Deskew calibration— (DDR4, QDR-IV, and LPDDR3 only)
  - (DDR4) Uses the DDR4 address and command parity feature. The FPGA sends the address and command parity bit, and the DDR4 memory device responds with an alert signal if the parity bit is detected. The alert signal from the memory device tells the FPGA that the parity bit was received.  
Deskew calibration requires use of the PAR/ALERT# pins, so you should not omit these pins from your design. One limitation of deskew calibration is that it cannot deskew ODT and CKE pins.
  - (QDR-IV) Uses the QDR-IV loopback mode. The FPGA sends address and command signals, and the memory device sends back the address and command signals which it captures, via the read data pins. The returned signals indicate to the FPGA what the memory device has captured. Deskew calibration can deskew all synchronous address and command signals.
  - (LPDDR3) Uses the LPDDR3 CA training mode. The FPGA sends signals onto the LPDDR3 CA bus, and the memory device sends back those signals that it captures, via the DQ pins. The returned signals indicate to the FPGA what the memory device has captured. Deskew calibration can deskew all signals on the CA bus. The remaining command signals (CS, CKE, and ODT) are calibrated based on the average of the deskewed CA bus.



## Read Calibration

- DQSen calibration— (DDR3, DDR4, LPDDR3, RLDRAMx and QDRx) DQSen calibration occurs before Read deskew, therefore only a single DQ bit is required to pass in order to achieve a successful read pass.
  - (DDR3, DDR4, and LPDDR3) The DQSen calibration algorithm searches the DQS preamble using a hardware state machine. The algorithm sends many back-to-back reads with a one clock cycle gap between. The hardware state machine searches for the DQS gap while sweeping DQSen delay values. The algorithm then increments the VFIFO value, and repeats the process until a pattern is found. The process is then repeated for all other read DQS groups.
  - (RLDRAMx and QDRx) The DQSen calibration algorithm does not use a hardware state machine; rather, it calibrates cycle-level delays using software and subcycle delays using DQS tracking hardware. The algorithm requires good data in memory, and therefore relies on guaranteed writes. (Writing a burst of 0s to one location, and a burst of 1s to another; back-to-back reads from these two locations are used for read calibration.)

The algorithm enables DQS tracking to calibrate the phase component of DQS enable, and then issues a guaranteed write, followed by back-to-back reads. The algorithm sweeps DQSen values cycle by cycle until the read operation succeeds. The process is then repeated for all other read groups.
- Deskew calibration— Read deskew calibration is performed before write leveling, and must be performed at least twice: once before write calibration, using simple data patterns from guaranteed writes, and again after write calibration, using complex data patterns.

The deskew calibration algorithm performs a guaranteed write, and then sweeps `dqs_in` delay values from low to high, to find the right edge of the read window. The algorithm then sweeps `dq_in` delay values low to high, to find the left edge of the read window. Updated `dqs_in` and `dq_in` delay values are then applied to center the read window. The algorithm then repeats the process for all data pins.
- Vref-In calibration— Read Vref-In calibration begins by programming `Vref-In` with an arbitrary value. The algorithm then sweeps the `Vref-In` value from the starting value to both ends, and measures the read window for each value. The algorithm selects the `Vref-In` value which provides the maximum read window.
- LFIFO calibration— Read LFIFO calibration normalizes read delays between groups. The PHY must present all data to the controller as a single data bus. The LFIFO latency should be large enough for the slowest read data group, and large enough to allow proper synchronization across FIFOs.

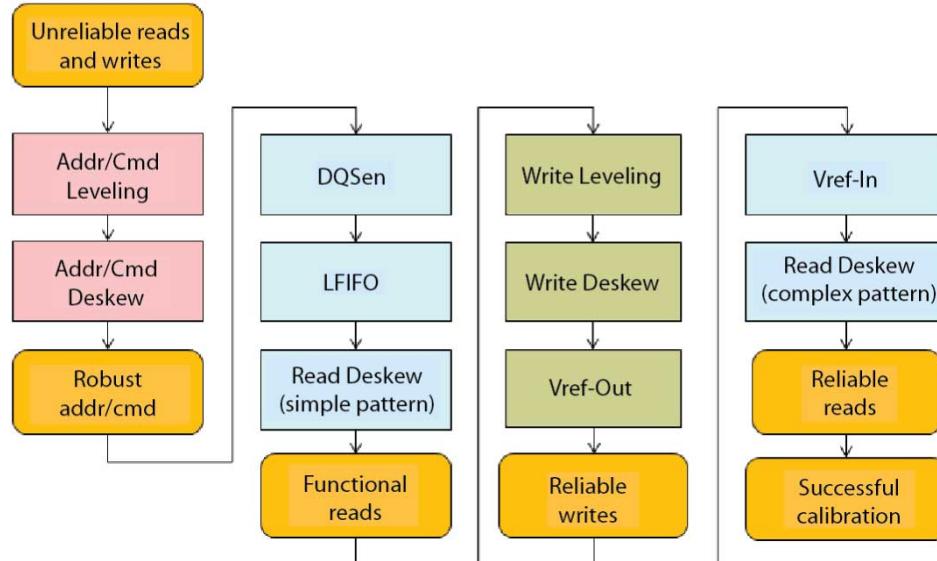
## Write Calibration

- Leveling calibration— Write leveling calibration aligns the write strobe and clock to the memory clock, to compensate for skews. In general, leveling calibration tries a variety of delay values to determine the edges of the write window, and then selects an appropriate value to center the window. The details of the algorithm vary, depending on the memory protocol.
  - (DDR<sub>x</sub>, LPDDR3) Write leveling occurs before write deskew, therefore only one successful DQ bit is required to register a pass. Write leveling staggers the DQ bus to ensure that at least one DQ bit falls within the valid write window.
  - (RLDRAM<sub>x</sub>) Optimizes for the CK versus DK relationship.
  - (QDR-IV) Optimizes for the CK versus DK relationship. Is covered by address and command deskew using the loopback mode.
  - (QDR II/II+/Xtreme) The K clock is the only clock, therefore write leveling is not required.
- Deskew calibration— Performs per-bit deskew of write data relative to the write strobe and clock. Write deskew calibration does not change dqs\_out delays; the write clock is aligned to the CK clock during write leveling.
- VREF-Out calibration— (DDR4) Calibrates the VREF level at the memory device. The VREF-Out calibration algorithm is similar to the VREF-In calibration algorithm.

### 2.9.4. Stratix 10 Calibration Flowchart

The following flowchart illustrates the Stratix 10 calibration flow.

**Figure 49. Calibration Flowchart**





## 2.10. Stratix 10 EMIF and SmartVID

Stratix 10 EMIF IP can be used with the SmartVID voltage management system, to achieve reduced power consumption.

The SmartVID controller allows the FPGA to operate at a reduced  $V_{CC}$ , while maintaining performance. Because the SmartVID controller can adjust  $V_{CC}$  up or down in response to power requirements and temperature, it can have an impact on external memory interface performance. When used with the SmartVID controller, the EMIF IP implements a handshake protocol to ensure that EMIF calibration does not begin until after voltage adjustment has completed.

In extended speed grade devices, voltage adjustment occurs once when the FPGA is powered up, and no further voltage adjustments occur. The external memory calibration occurs after this initial voltage adjustment is completed. EMIF specifications are expected to be slightly lower in extended speed grade devices using SmartVID, than in devices not using SmartVID.

In industrial speed grade devices, voltage adjustment occurs at power up, and may also occur during operation, in response to temperature changes. External memory interface calibration does not occur until after the initial voltage adjustment at power up. However, the external memory interface is not recalibrated in response to subsequent voltage adjustments that occur during operation. As a result, EMIF specifications for industrial speed grade devices using SmartVID are expected to be lower than for extended speed grade devices.

### Using Stratix 10 EMIF IP with SmartVID

To employ Stratix 10 EMIF IP with SmartVID, follow these steps:

1. Ensure that the Quartus Prime project and Qsys system are configured to use VID components. This step exposes the `vid_cal_done_persist` interface on instantiated EMIF IP, which is required for communicating with the SmartVID controller.
2. Instantiate the SmartVID controller, using an I/O PLL IP core to drive the 125MHz `vid_clk` and the 25MHz `jtag_core_clk` inputs of the Smart VID controller.

*Note:* Do not connect the `emif_usr_clk` signal to either the `vid_clk` or `jtag_core_clk` inputs. Doing so would hold both the EMIF IP and the SmartVID controller in a perpetual reset condition.

3. Instantiate the Stratix 10 EMIF IP.
4. Connect the `vid_cal_done_persist` signal from the EMIF IP with the `cal_done_persistent` signal on the SmartVID controller. This connection enables handshaking between the EMIF IP and the SmartVID controller, which allows the EMIF IP to delay memory calibration until after voltage levels are stabilized.

*Note:* The EMIF `vid_cal_done_persist` interface becomes available only when a VID-enabled device is selected.

## 2.11. Stratix 10 Hard Memory Controller Rate Conversion Feature

The hard memory controller's rate conversion feature allows the hard memory controller and PHY to run at half-rate, even though user logic is configured to run at quarter-rate.



To facilitate timing closure, you may choose to clock your core user logic at quarter-rate, resulting in easier timing closure at the expense of increased area and latency. To improve efficiency and help reduce overall latency, you can run the hard memory controller and PHY at half rate.

The rate conversion feature converts traffic from the FPGA core to the hard memory controller from quarter-rate to half-rate, and traffic from the hard memory controller to the FPGA core from half-rate to quarter-rate. From the perspective of user logic inside the FPGA core, the effect is the same as if the hard memory controller were running at quarter-rate.

The rate conversion feature is enabled automatically during IP generation whenever all of the following conditions are met:

- The hard memory controller is in use.
- User logic runs at quarter-rate.
- The interface targets either an ES2 or production device.
- Running the hard memory controller at half-rate does not exceed the fMax specification of the hard memory controller and hard PHY.

When the rate conversion feature is enabled, you should see the following info message displayed in the IP generation GUI:

PHY and controller running at 2x the frequency of user logic for improved efficiency.

## 2.12. Differences Between User-Requested Reset in Stratix 10 versus Arria 10

The following table highlights differences between the user-requested reset mechanism in the Arria 10 EMIF IP and the Stratix 10 EMIF IP.

**Table 27.**

|  | Arria 10   | Stratix 10  |
|--|--|---|
| Reset-related signals                  | global_reset_n   | local_reset_req<br>local_reset_done   |
| When can user logic request a reset?   | Any time after the FPGA enters user mode.  | local_reset_req has effect only when local_reset_done is high.<br>After device power-on, the local_reset_done signal transitions high upon completion of the first calibration, whether the calibration is successful or not.   |
| Is user-requested reset a requirement? | A user-requested reset is typically required to ensure the memory interface begins from a known state. | A user-requested reset is optional. The IOSSM (which is part of the device's CNOC) automatically ensures that the memory interface begins from a known state as part of the device power-on sequence. A user-requested reset is necessarily only if the user logic must explicitly reset a memory interface after the device power-on sequence. |

*continued...*



|   | <b>Arria 10</b>   | <b>Stratix 10</b>  |
|---|---|--|
| When does a user-requested reset actually happen?           | As soon as <code>global_reset_n</code> is driven low by user logic.   | A reset request is handled by the IOSSM. If the IOSSM receives a reset request from multiple interfaces within the same I/O column, it must serialize the reset sequence of the individual interfaces. You should avoid making assumptions on when the reset sequence will begin after a request is issued.  |
| Timing requirement and triggering mechanism.                | <code>global_reset_n</code> is an asynchronous, active-low reset signal. Reset assertion and de-assertion is level-triggered.           | Reset request is sent by transitioning the <code>local_reset_req</code> signal from low to high, then keeping the signal at the high state for a minimum of 2 EMIF core clock cycles, then transitioning the signal from high to low.<br><code>local_reset_req</code> is asynchronous in that there is no setup/hold timing to meet, but it must meet the minimum pulse width requirement of 2 EMIF core clock cycles. |
| How long can an external memory interface be kept in reset? | The interface is kept in reset for as long as <code>global_reset_n</code> is driven low.  | It is not possible to keep an external memory interface in reset indefinitely. Asserting <code>local_reset_req</code> high continuously has no effect as a reset request is completed by a full 0->1->0 pulse.   |
| Delaying initial calibration.                               | Initial calibration can be delayed for as long as desired by driving <code>global_reset_n</code> immediately after FPGA power-up.       | Initial calibration cannot be skipped. The <code>local_reset_done</code> signal is driven high only after initial calibration has completed.   |
| Reset scope (within an external memory interface).          | All circuits involved in EMIF operations are reset.   | Only circuits that are required to restore EMIF to power-up state are reset. Excluded from the reset sequence are the IOSSM, the IOPLL(s), the DLL(s), and the CPA.  |
| Reset scope (within an I/O column).                         | <code>global_reset_n</code> is a column-wide reset. It is not possible to reset a subset of the memory interfaces within an I/O column. | <code>local_reset_req</code> is a per-interface reset.   |

## 2.12.1. Method for Initiating a User-requested Reset

### Step 1 - Precondition

Before asserting `local_reset_req`, user logic must ensure that the `local_reset_done` signal is high.

As part of the device power-on sequence, the `local_reset_done` signal automatically transitions to high upon the completion of the interface calibration sequence, regardless of whether calibration is successful or not.

**Note:** When targeting a group of interfaces that share the same core clocks, user logic must ensure that the `local_reset_done` signal of every interface is high.



## Step 2 - Reset Request

After the pre-condition is satisfied, user logic can send a reset request by driving the local\_cal\_req signal from low to high and then low again (that is, by sending a pulse of 1).

- The 0-to-1 and 1-to-0 transitions need not happen in relation to any clock edges (that is, they can occur asynchronously); however, the pulse must meet a minimum pulse width of at least 2 EMIF core clock cycles. For example, if the emif\_usr\_clk has a period of 4ns, then the local\_reset\_req pulse must last at least 8ns (that is, two emif\_usr\_clk periods).
- The reset request is considered complete only after the 1-to-0 transition. The EMIF IP does not initiate the reset sequence when the local\_reset\_req is simply held high.
- Additional pulses to local\_reset\_req are ignored until the reset sequence is completed.

### Optional - Detecting local\_reset\_done deassertion and assertion

If you want, you can monitor the status of the local\_reset\_done signal to explicitly detect the status of the reset sequence.

- After the EMIF IP receives a reset request, it deasserts the local\_reset\_done signal. After initial power-up calibration, local\_reset\_done is de-asserted only in response to a user-requested reset. The reset sequence is imminent when local\_reset\_done has transitioned to low, although the exact timing depends on the current state of the I/O Subsystem Manager (IOSSM). As part of the EMIF reset sequence, the core reset signal (emif\_usr\_reset\_n, afi\_reset\_n) is driven low. Do not use a register reset by the core reset signal to sample local\_reset\_done.
- After the reset sequence has completed, local\_reset\_done is driven high again. local\_reset\_done being driven high indicates the completion of the reset sequence and the readiness to accept a new reset request; however, it does not imply that calibration was successful or that the hard memory controller is ready to accept requests. For these purposes, user logic must check signals such as afi\_cal\_success, afi\_cal\_fail, and amm\_ready.

## 2.13. Compiling Stratix 10 EMIF IP with the Quartus Prime Software

### 2.13.1. Instantiating the Stratix 10 EMIF IP

Depending on your work flow, you may instantiate your IP with Qsys or with the IP Catalog.

#### Instantiating with Qsys

If you instantiate your IP as part of a system in Qsys, follow the Qsys documentation for information on instantiating the IP in a Quartus Prime project.



### Instantiating with the IP Catalog

If you generated your IP with the IP Catalog, you must add the Quartus Prime IP file (.qip) to your Quartus Prime project. The .qip file identifies the names and locations of the files that compose the IP. After you add the .qip file to your project, you can instantiate the memory interface in the RTL.

## 2.13.2. Setting I/O Assignments in Stratix 10 EMIF IP

The .qip file contains the I/O standard and I/O termination assignments required by the memory interface pins for proper operation. The assignment values are based on input that you provide during generation.

Unlike earlier device families, for Stratix 10 EMIF IP you do not need to run a `<instance_name>_pin_assignments.tcl` script to add the assignments into the Quartus Prime Settings File (.qsf). The system reads and applies the assignments from the .qip file during every compilation, regardless of how you name the memory interface pins in the top-level design component. No new assignments are created in the project's .qsf file during compilation.

Note that I/O assignments in the .qsf file must specify the names of your top-level pins as target (-to), and you must not include the -entity or -library options.

Consult the generated .qip file for the set of I/O assignments that are provided with the IP.

### Changing I/O Assignments

You should not make changes to the generated .qip file, because any changes are overwritten and lost when you regenerate the IP. If you want to override an assignment made in the .qip file, add the desired assignment to the project's .qsf file. Assignments in the .qsf file always take precedence over assignments in the .qip file.

## 2.14. Debugging Stratix 10 EMIF IP

You can debug hardware failures by connecting to the EMIF Debug Toolkit or by exporting an Avalon-MM slave port, from which you can access information gathered during calibration. You can also connect to this port to mask ranks and to request recalibration.

You can access the exported Avalon-MM port in two ways:

- Via the External Memory Interface Debug Toolkit
- Via On-Chip Debug (core logic on the FPGA)

### 2.14.1. External Memory Interface Debug Toolkit

The External Memory Interface Debug Toolkit provides access to data collected by the Nios II sequencer during memory calibration, and allows you to perform certain tasks.



The External Memory Interface Debug Toolkit provides access to data including the following:

- General interface information, such as protocol and interface width
- Calibration results per group, including pass/fail status, failure stage, and delay settings

You can also perform the following tasks:

- Mask ranks from calibration (you might do this to skip specific ranks)
- Request recalibration of the interface

### 2.14.2. On-Chip Debug for Stratix 10

The On-Chip Debug feature allows user logic to access the same debug capabilities as the External Memory Interface Toolkit. You can use On-Chip Debug to monitor the calibration results of an external memory interface, without a connected computer.

To use On-Chip Debug, you need a C header file which is provided as part of the external memory interface IP. The C header file defines data structures that contain calibration data, and definitions of the commands that can be sent to the memory interface.

The On-Chip Debug feature accesses the data structures through the Avalon-MM port that is exposed by the EMIF IP when you turn on debugging features.

### 2.14.3. Configuring Your EMIF IP for Use with the Debug Toolkit

The Stratix 10 EMIF Debug Interface IP core contains the access point through which the EMIF Debug Toolkit reads calibration data collected by the Nios II sequencer.

#### Connecting an EMIF IP Core to a Stratix 10 EMIF Debug Interface

For the EMIF Debug Toolkit to access the calibration data for a Stratix 10 EMIF IP core, you must connect one of the EMIF cores in each I/O column to a Stratix 10 EMIF Debug Interface IP core. Subsequent EMIF IP cores in the same column must connect in a daisy chain to the first.

There are two ways that you can add the Stratix 10 EMIF Debug Interface IP core to your design:

- When you generate your EMIF IP core, on the **Diagnostics** tab, select **Add EMIF Debug Interface** for the **EMIF Debug Toolkit/On-Chip Debug Port**; you do not have to separately instantiate a Stratix 10 EMIF Debug Interface core. This method does not export an Avalon-MM slave port. You can use this method if you require only EMIF Debug Toolkit access to this I/O column; that is, if you do not require On-Chip Debug Port access, or PHYLITE reconfiguration access.
- When you generate your EMIF IP core, on the **Diagnostics** tab, select **Export** for the **EMIF Debug Toolkit/On-Chip Debug Port**. Then, separately instantiate a Stratix 10 EMIF Debug Interface core and connect its `to_iossm` interface to the `cal_debug` interface on the EMIF IP core. This method is appropriate if you want to also have On-Chip Debug Port access to this I/O column, or PHYLITE reconfiguration access.



For each of the above methods, you must assign a unique interface ID for each external memory interface in the I/O column, to identify that interface in the Debug Toolkit. You can assign an interface ID using the dropdown list that appears when you enable the **Debug Toolkit/On-Chip Debug Port** option.

### Daisy-Chaining Additional EMIF IP Cores for Debugging

After you have connected a Stratix 10 EMIF Debug Interface to one of the EMIF IP cores in an I/O column, you must then connect subsequent EMIF IP cores in that column in a daisy-chain manner. If you don't require debug capabilities for a particular EMIF IP core, you do not have to connect that core to the daisy chain.

To create a daisy chain of EMIF IP cores, follow these steps:

1. On the first EMIF IP core, select **Enable Daisy-Chaining for EMIF Debug Toolkit/On-Chip Debug Port** to create an Avalon-MM interface called cal\_debug\_out.
2. On the second EMIF IP core, select **Export** as the **EMIF Debug Toolkit/On-Chip Debug Port** mode, to export an Avalon-MM interface called cal\_debug.
3. Connect the cal\_debug\_out interface of the first EMIF core to the cal\_debug interface of the second EMIF core.
4. To connect more EMIF cores to the daisy chain, select the **Enable Daisy-Chaining for EMIF Debug Toolkit/On-Chip Debug Port** option on the second core, connect it to the next core using the **Export** option as described above. Repeat the process for subsequent EMIF cores.

If you place any PHYLite cores with dynamic reconfiguration enabled into the same I/O column as an EMIF IP core, you should instantiate and connect the PHYLite cores in a similar way. See the *Altera PHYLite for Memory Megafunction User Guide* for more information.

### Related Information

[Altera PHYLite for Parallel Interfaces IP Core User Guide](#)

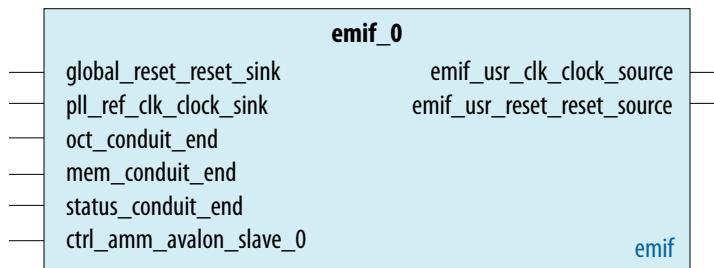
## 2.14.4. Stratix 10 EMIF Debugging Examples

The following sections provide examples of debugging a single external memory interface, and of adding additional EMIF instances to an I/O column.

### Debugging a Single External Memory Interface

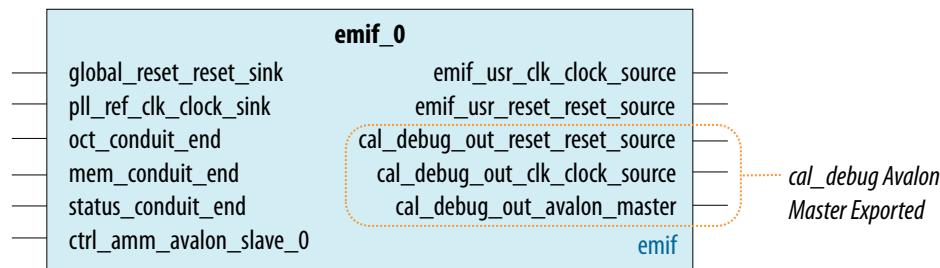
1. Under **EMIF Debug Toolkit/On-Chip Debug Port**, select **Add EMIF Debug Interface**.  
(If you want to use the On-Chip Debug Port instead of the EMIF Debug Toolkit, select **Export** instead.)

**Figure 50. EMIF With Debug Interface Added (No Additional Ports)**



2. If you want to connect additional EMIF or PHYLite components in this I/O column, select **Enable Daisy Chaining for EMIF Debug Toolkit/On-Chip Debug Port**.

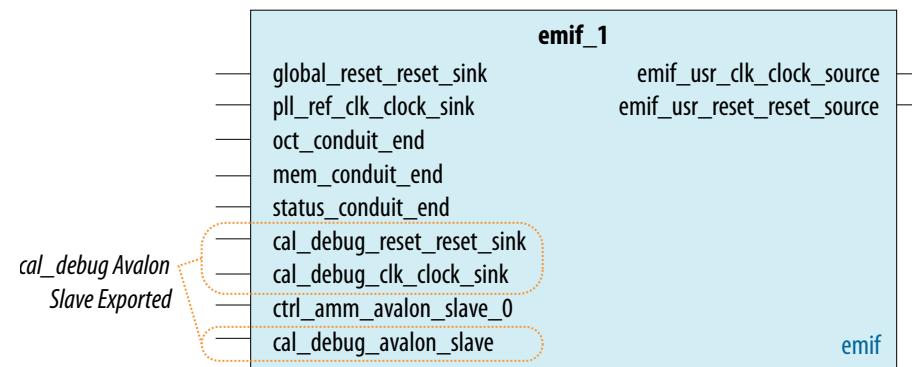
**Figure 51. EMIF With cal\_debug Avalon Master Exported**



#### Adding Additional EMIF Instances to an I/O Column

1. Under **EMIF Debug Toolkit/On-Chip Debug Port**, select **Export**.

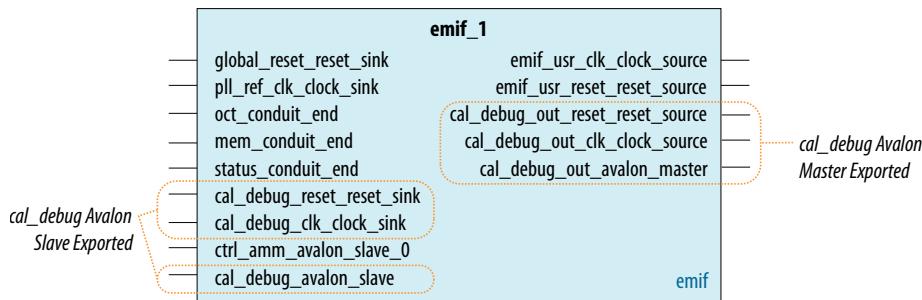
**Figure 52. EMIF With cal\_debug Avalon Slave Exported**



2. Specify a unique interface ID for this EMIF instance.
3. If you want to connect additional EMIF or PHYLite components in this I/O column, select **Enable Daisy Chaining for EMIF Debug Toolkit/On-Chip Debug Port**.

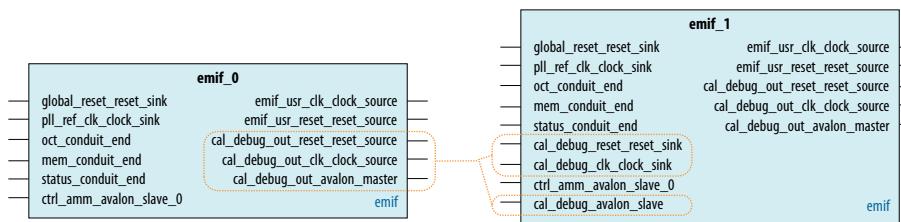


**Figure 53. EMIF With Both cal\_debug Master and Slave Exported**



4. Connect the cal\_debug Avalon Master, clock, and reset interfaces of the previous component to the cal\_debug Avalon Slave, clock, and reset interfaces of this component.

**Figure 54. EMIF Components Connected**



## 2.15. Stratix 10 EMIF for Hard Processor Subsystem

The Stratix 10 EMIF IP can enable the Stratix 10 Hard Processor Subsystem (HPS) to access external DRAM memory devices.

To enable connectivity between the Stratix 10 HPS and the Stratix 10 EMIF IP, you must create and configure an instance of the Stratix 10 External Memory Interface for HPS IP core, and use Qsys to connect it to the Stratix 10 Hard Processor Subsystem instance in your system.

### Supported Modes

The Stratix 10 Hard Processor Subsystem is compatible with the following external memory configurations:

|  |   |
|--|---|
| Protocol                                     | DDR3, DDR4, LPDDR3                                    |
| Maximum memory clock frequency               | DDR3: 1.067 GHz<br>DDR4: 1.333 GHz<br>LPDDR3: 800 MHz |
| Configuration                                | Hard PHY with hard memory controller                  |
| Clock rate of PHY and hard memory controller | Half-rate   |
| Data width (without ECC)                     | 16-bit, 32-bit, 64-bit                                |
| Data width (with ECC)                        | 24-bit, 40-bit, 72-bit                                |
| DQ width per group                           | x8  |

*continued...*



|   |                                |
|---|--------------------------------|
| Maximum number of I/O lanes for address/command | 3                              |
| Memory format                                   | Discrete, UDIMM, SODIMM, RDIMM |
| Ranks / CS# width                               | Up to 2                        |

## 2.15.1. Restrictions on I/O Bank Usage for Stratix 10 EMIF IP with HPS

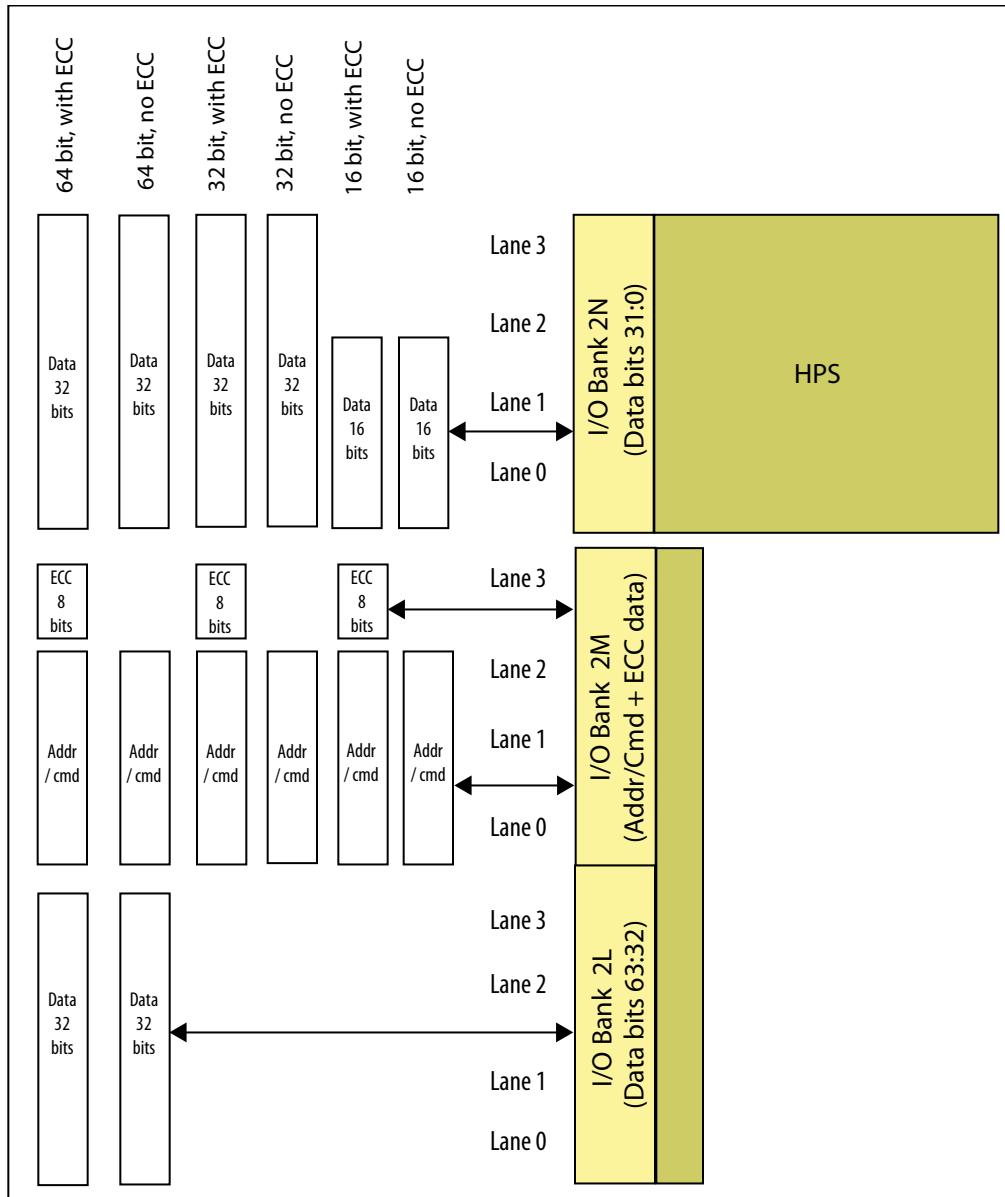
You can use only certain Stratix 10 I/O banks to implement Stratix 10 EMIF IP with the Stratix 10 Hard Processor Subsystem.

The restrictions on I/O bank usage result from the Stratix 10 HPS having hard-wired connections to the EMIF circuits in the I/O banks closest to the HPS. For any given EMIF configuration, the pin-out of the EMIF-to-HPS interface is fixed.



The following diagram illustrates the use of I/O banks and lanes for various EMIF-HPS data widths:

**Figure 55. Stratix 10 External Memory Interfaces I/O Bank and Lanes Usage**



The HPS EMIF uses the closest located external memory interfaces I/O banks to connect to SDRAM. These banks include:

- Bank 2N—used for data I/Os (Data bits 31:0)
- Bank 2M—used for address, command and ECC data I/Os
- Bank 2L—used for data I/Os (Data bits 63:32)



If no HPS EMIF is used in a system, the entire HPS EMIF bank can be used as FPGA GPIO. If there is a HPS EMIF in a system, the unused HPS EMIF pins can be used as FPGA general I/O with restrictions:

- Bank 2M:
  - Lane 3 is used for SDRAM ECC data. Unused pins in lane 3 can be used as FPGA inputs only.
  - Lanes 2, 1, and 0 are used for SDRAM address and command. Unused pins in these lanes can be used as FPGA inputs or outputs.
- Bank 2N and Bank 2L :
  - Lanes 3, 2, 1, and 0 are used for data bits.
  - With 64-bit data widths, unused pins in these banks can be used as FPGA inputs only.
  - With 32-bit data widths, unused pins in Bank 2N can be used as FPGA inputs only. Unused pins for Bank 2L can be used as FPGA inputs or outputs.
  - With 16-bit data widths, Quartus® Prime assigns lane 0 and lane 1 as data lanes in bank 2N. Unused pins in lane 0 and lane 1 can be used as FPGA inputs only. The other two lanes are available to use as FPGA inputs or outputs.

By default, the Stratix 10 External Memory Interface for HPS IP core together with the Quartus Prime Fitter automatically implement the correct pin-out for HPS EMIF without you having to apply additional constraints. If you must modify the default pin-out for any reason, you must adhere to the following requirements, which are specific to HPS EMIF:

1. Within a single data lane (which implements a single x8 DQS group):
  - DQ pins must use pins at indices 1, 2, 3, 6, 7, 8, 9, 10. You may swap the locations between the DQ bits (that is, you may swap location of DQ[0] and DQ[3]) so long as the resulting pin-out uses pins at these indices only.
  - DM/DBI pin must use pin at index 11. There is no flexibility.
  - DQS/DQS# must use pins at index 4 and 5. There is no flexibility.
2. Assignment of data lanes must be as illustrated in the above figure. You are allowed to swap the locations of entire byte lanes (that is, you may swap locations of byte 0 and byte 3) so long as the resulting pin-out uses only the lanes permitted by your HPS EMIF configuration, as shown in the above figure.
3. You must not change placement of the address and command pins from the default.
4. You may place the alert# pin at any available pin location in either a data lane or an address and command lane.

To override the default generated pin assignments, comment out the relevant HPS\_LOCATION assignments in the .qip file, and add your own location assignments (using set\_location\_assignment) in the .qsf file.



## 2.16. Stratix 10 EMIF Ping Pong PHY

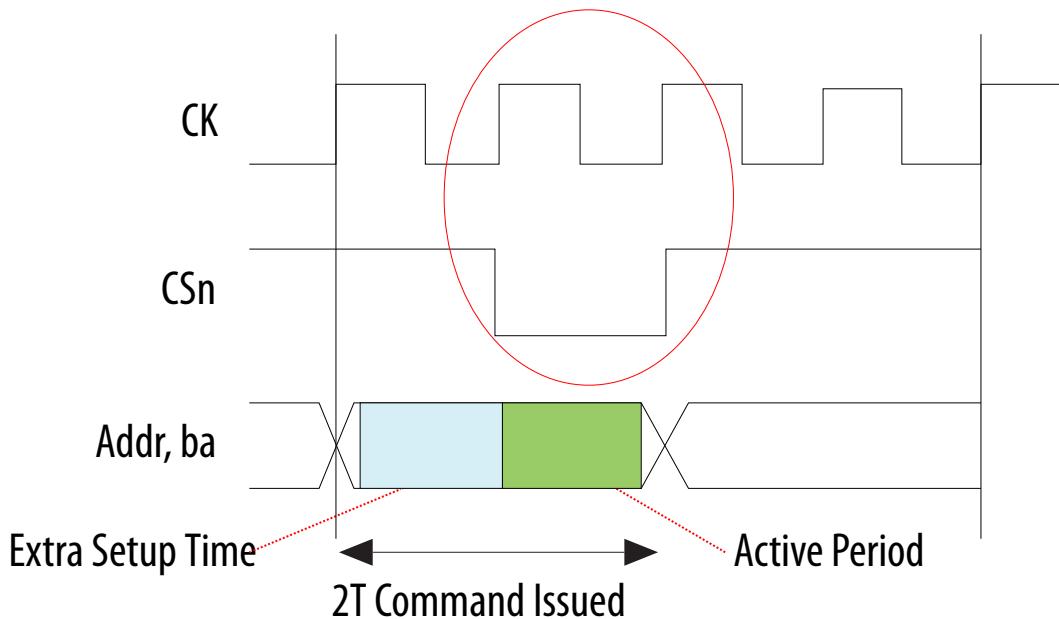
Ping Pong PHY allows two memory interfaces to share the address and command bus through time multiplexing. Compared to having two independent interfaces that allocate address and command lanes separately, Ping Pong PHY achieves the same throughput with fewer resources, by sharing the address and command lanes.

In Stratix 10 EMIF, Ping Pong PHY supports both half-rate and quarter-rate interfaces for DDR3, and quarter-rate for DDR4.

### 2.16.1. Stratix 10 Ping Pong PHY Feature Description

Conventionally, the address and command buses of a DDR3 or DDR4 half-rate or quarter-rate interface use 2T time—meaning that commands are issued for two full-rate clock cycles, as illustrated below.

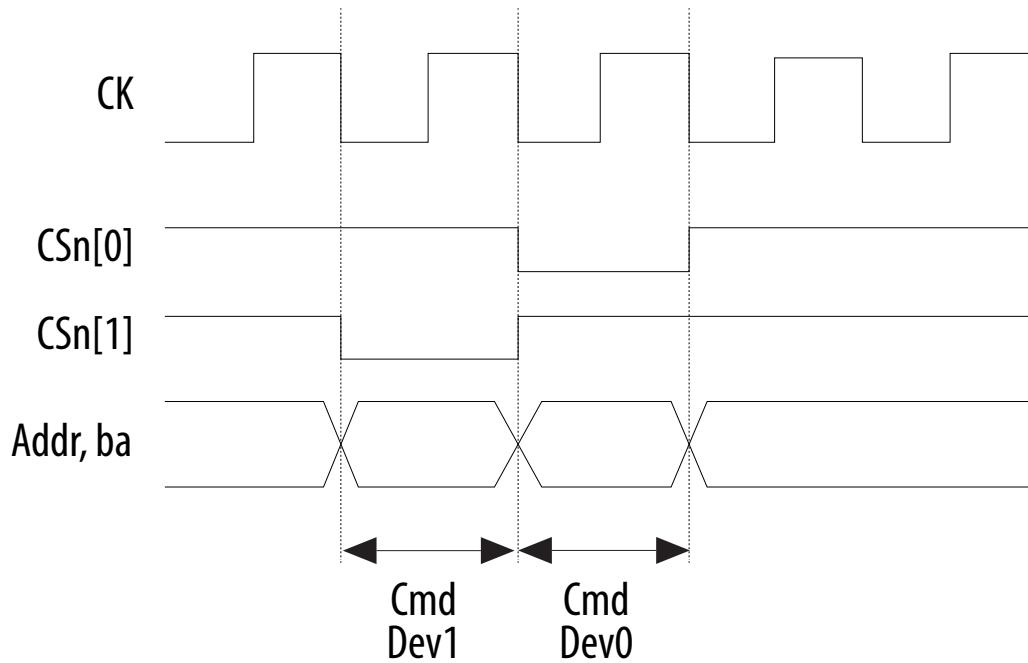
**Figure 56. 2T Command Timing**



With the Ping Pong PHY, address and command signals from two independent controllers are multiplexed onto shared buses by delaying one of the controller outputs by one full-rate clock cycle. The result is 1T timing, with a new command being issued on each full-rate clock cycle. The following figure shows address and command timing for the Ping Pong PHY.

The command signals CS, ODT, and CKE have two signals (one for ping and one for pong); the other address and command signals are shared.

**Figure 57. 1T Command Timing Use by Ping Pong PHY**



### 2.16.2. Stratix 10 Ping Pong PHY Architecture

In Stratix 10 EMIF, the Ping Pong PHY feature can be enabled only with the hard memory controller, where two hard memory controllers are instantiated—one for the primary interface and one for the secondary interface.

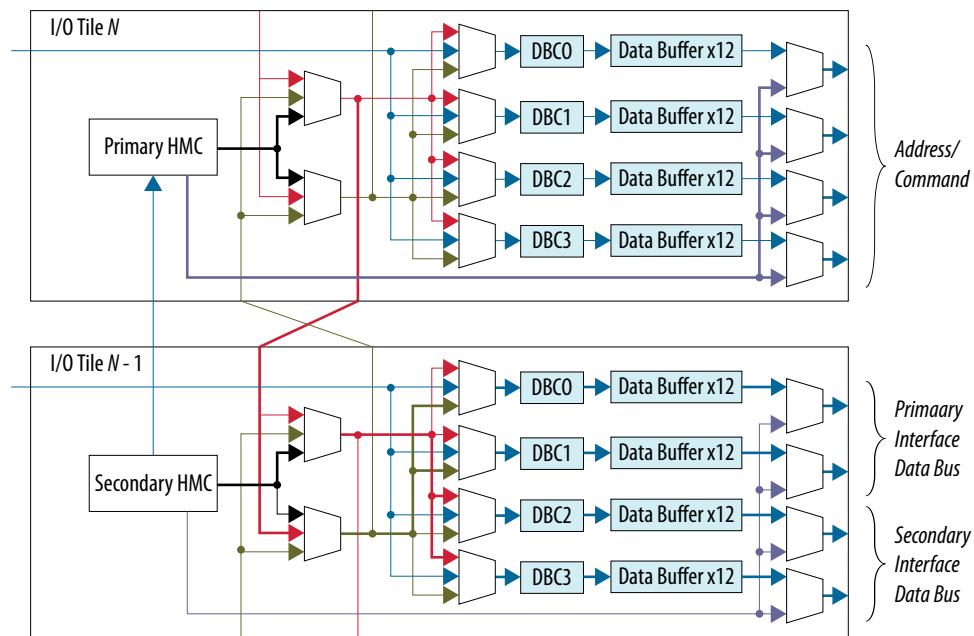
The hard memory controller I/O bank of the primary interface is used for address and command and is always adjacent and above the hard memory controller bank of the secondary interface. All four lanes of the primary hard memory controller bank are used for address and command.

The following example shows a 2x16 Ping Pong PHY bank-lane configuration. The upper bank (I/O bank N) is the address and command bank, which serves both the primary and secondary interfaces. The primary hard memory controller is linked to the secondary interface by the Ping Pong bus. The lower bank (I/O bank N-1) is the secondary interface bank, which carries the data buses for both primary and secondary interfaces. In the 2x16 case a total of four I/O banks are required for data, hence two banks in total are sufficient for the implementation.

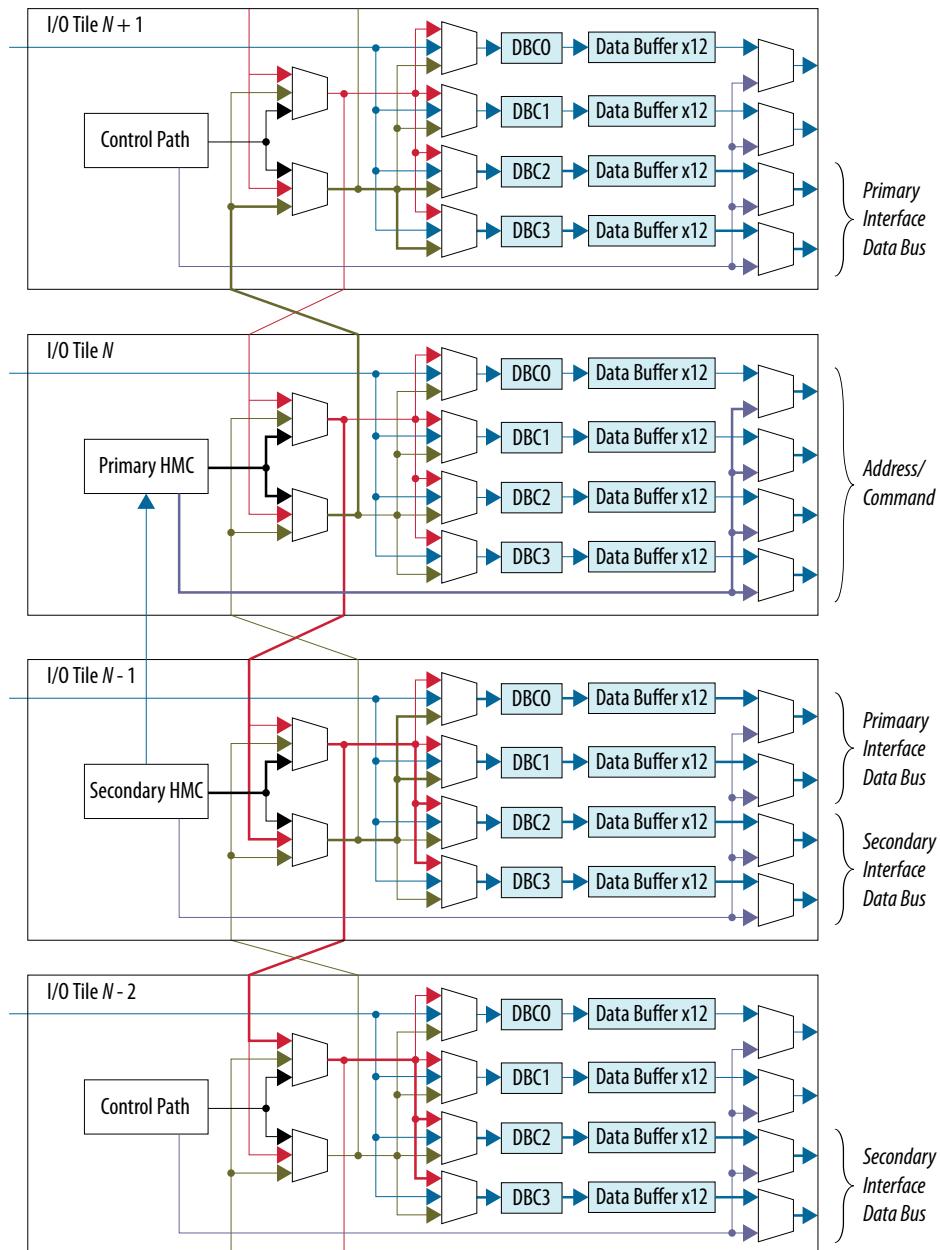
The data for the primary interface is routed down to the top two lanes of the secondary I/O bank, and the data for the secondary interface is routed to the bottom two lanes of the secondary I/O bank.



**Figure 58. 2x16 Ping Pong PHY I/O Bank-Lane Configuration**



A 2x32 interface can be implemented similarly, with the additional data lanes placed above and below the primary and secondary I/O banks, such that primary data lanes are placed above the primary bank and secondary data lanes are placed below the secondary bank.

**Figure 59. 2x32 Ping Pong PHY I/O Bank-Lane Configuration.**


### 2.16.3. Stratix 10 Ping Pong PHY Limitations

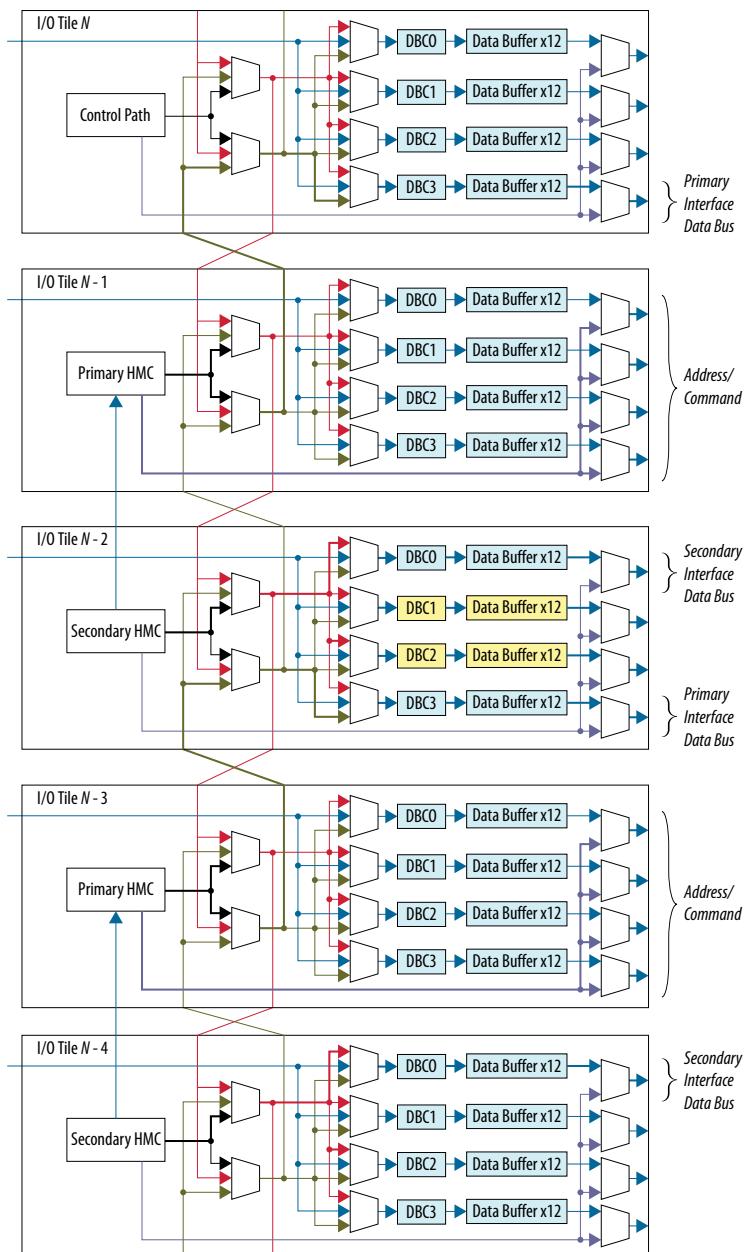
Ping Pong PHY supports up to two ranks per memory interface. In addition, the maximum data width is x72, which is half the maximum width of x144 for a single interface.



Ping Pong PHY uses all lanes of the address and command I/O bank as address and command. For information on the pin allocations of the DDR3 and DDR4 address and command I/O bank, refer to *DDR3 Scheme 1* and *DDR4 Scheme 3*, in *External Memory Interface Pin Information for Stratix 10 Devices*, on [www.altera.com](http://www.altera.com).

An additional limitation is that I/O lanes may be left unused when you instantiate multiple pairs of Ping Pong PHY interfaces. The following diagram shows two pairs of x8 Pin Pong controllers (a total of 4 interfaces). Lanes highlighted in yellow are not driven by any memory interfaces (unused lanes and pins can still serve as general purpose I/Os). Even with some I/O lanes left unused, the Ping Pong PHY approach is still beneficial in terms of resource usage, compared to independent interfaces. Memory widths of 24 bits and 40 bits have a similar situation, while 16 bit, 32 bit, and 64 bit memory widths do not suffer this limitation.

**Figure 60. Two Pairs of x8 Pin-Pong PHY Controllers**



## 2.16.4. Stratix 10 Ping Pong PHY Calibration

A Ping Pong PHY interface is calibrated as a regular interface of double width.

Calibration of a Ping Pong PHY interface incorporates two sequencers, one on the primary hard memory controller I/O bank, and one on the secondary hard memory controller I/O bank. To ensure that the two sequencers issue instructions on the same memory clock cycle, the Nios II processor configures the sequencer on the primary hard memory controller to receive a token from the secondary interface, ignoring any



commands from the Avalon bus. Additional delays are programmed on the secondary interface to allow for the passing of the token from the sequencer on the secondary hard memory controller tile to the sequencer on the primary hard memory controller tile. During calibration, the Nios II processor assumes that commands are always issued from the sequencer on the primary hard memory controller I/O bank. After calibration, the Nios II processor adjusts the delays for use with the primary and secondary hard memory controllers.

## 2.16.5. Using the Ping Pong PHY

The following steps describe how to use the Ping Pong PHY for Stratix 10 EMIF.

1. Configure a single memory interface according to your requirements.
2. Select **Instantiate two controllers sharing a Ping Pong PHY** on the **General** tab in the parameter editor.

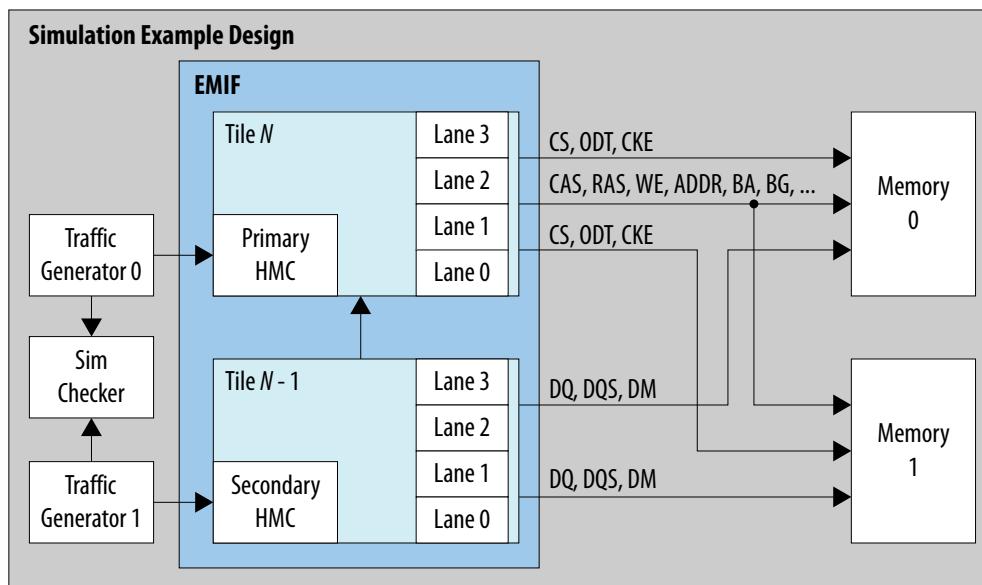
The Quartus Prime software replicates the interface, resulting in two memory controllers and a shared PHY. The system configures the I/O bank-lane structure, without further input from you.

## 2.16.6. Ping Pong PHY Simulation Example Design

The following figure illustrates a top-level block diagram of a generated Ping Pong PHY simulation example design, using two I/O banks.

Functionally, the IP interfaces with user traffic separately, as it would with two independent memory interfaces. You can also generate synthesizable example designs, where the external memory interface IP interfaces with a traffic generator.

**Figure 61. Ping Pong PHY Simulation Example Design**



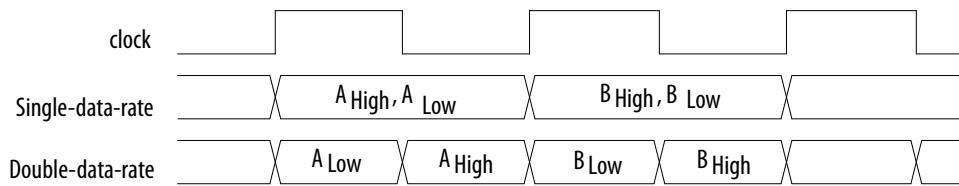
## 2.17. AFI 4.0 Specification

The Altera PHY interface (AFI) 4.0 defines communication between the controller and physical layer (PHY) in the external memory interface IP.

The AFI is a single-data-rate interface, meaning that data is transferred on the rising edge of each clock cycle. Most memory interfaces, however, operate at double-data-rate, transferring data on both the rising and falling edges of the clock signal. If the AFI interface is to directly control a double-data-rate signal, two single-data-rate bits must be transmitted on each clock cycle; the PHY then sends out one bit on the rising edge of the clock and one bit on the falling edge.

The AFI convention is to send the low part of the data first and the high part second, as shown in the following figure.

**Figure 62. Single Versus Double Data Rate Transfer**



### 2.17.1. Bus Width and AFI Ratio

In cases where the AFI clock frequency is one-half or one-quarter of the memory clock frequency, the AFI data must be twice or four times as wide, respectively, as the corresponding memory data. The ratio between AFI clock and memory clock frequencies is referred to as the AFI ratio. (A half-rate AFI interface has an AFI ratio of 2, while a quarter-rate interface has an AFI ratio of 4.)

In general, the width of the AFI signal depends on the following three factors:

- The size of the equivalent signal on the memory interface. For example, if `a[15:0]` is a DDR3 address input and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_addr` bus will be 16-bits wide.
- The data rate of the equivalent signal on the memory interface. For example, if `d[7:0]` is a double-data-rate QDR II input data bus and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_write_data` bus will be 16-bits wide.
- The AFI ratio. For example, if `cs_n` is a single-bit DDR3 chip select input and the AFI clock runs at half the speed of the memory interface, the equivalent `afi_cs_n` bus will be 2-bits wide.

The following formula summarizes the three factors described above:

```
AFI_width = memory_width * signal_rate * AFI_RATE_RATIO
```

**Note:** The above formula is a general rule, but not all signals obey it. For definite signal-size information, refer to the specific table.

### 2.17.2. AFI Parameters

The following tables list Altera PHY interface (AFI) parameters for AFI 4.0.

The parameters described in the following tables affect the width of AFI signal buses. Parameters prefixed by `MEM_IF_` refer to the signal size at the interface between the PHY and memory device.



**Table 28. Ratio Parameters**

| Parameter Name  | Description  |
|-----------------|--|
| AFI_RATE_RATIO  | The ratio between the AFI clock frequency and the memory clock frequency. For full-rate interfaces this value is 1, for half-rate interfaces the value is 2, and for quarter-rate interfaces the value is 4. |
| DATA_RATE_RATIO | The number of data bits transmitted per clock cycle. For single-data rate protocols this value is 1, and for double-data rate protocols this value is 2.   |
| ADDR_RATE_RATIO | The number of address bits transmitted per clock cycle. For single-data rate address protocols this value is 1, and for double-data rate address protocols this value is 2.                                  |

**Table 29. Memory Interface Parameters**

| Parameter Name         | Description  |
|------------------------|--|
| MEM_IF_ADDR_WIDTH      | The width of the address bus on the memory device(s).  |
| MEM_IF_BANKADDR_WIDTH  | The width of the bank address bus on the interface to the memory device(s). Typically, the $\log_2$ of the number of banks.                                |
| MEM_IF_CS_WIDTH        | The number of chip selects on the interface to the memory device(s).   |
| MEM_IF_WRITE_DQS_WIDTH | The number of DQS (or write clock) signals on the write interface. For example, the number of DQS groups.  |
| MEM_IF_CLK_PAIR_COUNT  | The number of CK/CK# pairs.  |
| MEM_IF_DQ_WIDTH        | The number of DQ signals on the interface to the memory device(s). For single-ended interfaces such as QDR II, this value is the number of D or Q signals. |
| MEM_IF_DM_WIDTH        | The number of data mask pins on the interface to the memory device(s).   |
| MEM_IF_READ_DQS_WIDTH  | The number of DQS signals on the read interface. For example, the number of DQS groups.  |

**Table 30. Derived AFI Parameters**

| Parameter Name      | Derivation Equation                                       |
|---------------------|---|
| AFI_ADDR_WIDTH      | MEM_IF_ADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO      |
| AFI_BANKADDR_WIDTH  | MEM_IF_BANKADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO  |
| AFI_CONTROL_WIDTH   | AFI_RATE_RATIO * ADDR_RATE_RATIO                          |
| AFI_CS_WIDTH        | MEM_IF_CS_WIDTH * AFI_RATE_RATIO                          |
| AFI_DM_WIDTH        | MEM_IF_DM_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO        |
| AFI_DQ_WIDTH        | MEM_IF_DQ_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO        |
| AFI_WRITE_DQS_WIDTH | MEM_IF_WRITE_DQS_WIDTH * AFI_RATE_RATIO                   |
| AFI_LAT_WIDTH       | 6   |
| AFI_RLAT_WIDTH      | AFI_LAT_WIDTH   |
| AFI_WLAT_WIDTH      | AFI_LAT_WIDTH * MEM_IF_WRITE_DQS_WIDTH                    |
| AFI_CLK_PAIR_COUNT  | MEM_IF_CLK_PAIR_COUNT                                     |
| AFI_WRANK_WIDTH     | Number of ranks * MEM_IF_WRITE_DQS_WIDTH * AFI_RATE_RATIO |
| AFI_RRANK_WIDTH     | Number of ranks * MEM_IF_READ_DQS_WIDTH * AFI_RATE_RATIO  |



### 2.17.3. AFI Signals

The following tables list Altera PHY interface (AFI) signals grouped according to their functions.

In each table, the **Direction** column denotes the direction of the signal relative to the PHY. For example, a signal defined as an output passes out of the PHY to the controller. The AFI specification does not include any bidirectional signals.

Not all signals are used for all protocols.

#### 2.17.3.1. AFI Clock and Reset Signals

The AFI interface provides up to two clock signals and an asynchronous reset signal.

**Table 31. Clock and Reset Signals**

| Signal Name  | Direction | Width | Description  |
|--------------|-----------|-------|--|
| afi_clk      | Output    | 1     | Clock with which all data exchanged on the AFI bus is synchronized. In general, this clock is referred to as full-rate, half-rate, or quarter-rate, depending on the ratio between the frequency of this clock and the frequency of the memory device clock. |
| afi_half_clk | Output    | 1     | Clock signal that runs at half the speed of the afi_clk. The controller uses this signal when the half-rate bridge feature is in use. This signal is optional.   |
| afi_reset_n  | Output    | 1     | Asynchronous reset output signal. You must synchronize this signal to the clock domain in which you use it.  |

#### 2.17.3.2. AFI Address and Command Signals

The address and command signals for AFI 4.0 encode read/write/configuration commands to send to the memory device. The address and command signals are single-data rate signals.

**Table 32. Address and Command Signals**

| Signal Name | Direction | Width               | Description  |
|-------------|-----------|---------------------|--|
| afi_addr    | Input     | AFI_ADDR_WIDTH      | Address or CA bus (LPDDR3 only). ADDR_RATE_RATIO is 2 for LPDDR3 CA bus.   |
| afi_bg      | Input     | AFI_BANKGROUP_WIDTH | Bank group (DDR4 only).  |
| afi_ba      | Input     | AFI_BANKADDR_WIDTH  | Bank address. (Not applicable for LPDDR3.)   |
| afi_cke     | Input     | AFI_CLK_EN_WIDTH    | Clock enable.  |
| afi_cs_n    | Input     | AFI_CS_WIDTH        | Chip select signal. (The number of chip selects may not match the number of ranks; for example, RDIMMs and LRDIMMs require a minimum of 2 chip select signals for both single-rank and dual-rank configurations. Consult your memory device data sheet for |

*continued...*



| Signal Name         | Direction | Width                 | Description  |
|---------------------|-----------|-----------------------|--|
|                     |           |                       | information about chip select signal width.) (Matches the number of ranks for LPDDR3.)   |
| afi_ras_n           | Input     | AFI_CONTROL_WIDTH     | RAS# (for DDR2 and DDR3 memory devices.)   |
| afi_we_n            | Input     | AFI_CONTROL_WIDTH     | WE# (for DDR2, DDR3, and RLDRAM II memory devices.)  |
| afi_rw_n            | Input     | AFI_CONTROL_WIDTH * 2 | RWA/B# (QDR-IV).   |
| afi_cas_n           | Input     | AFI_CONTROL_WIDTH     | CAS# (for DDR2 and DDR3 memory devices.)   |
| afi_act_n           | Input     | AFI_CONTROL_WIDTH     | ACT# (DDR4).   |
| afi_ref_n           | Input     | AFI_CONTROL_WIDTH     | REF# (for RLDRAM II memory devices.)   |
| afi_RST_n           | Input     | AFI_CONTROL_WIDTH     | RESET# (for DDR3 and DDR4 memory devices.)   |
| afi_odt             | Input     | AFI_CLK_EN_WIDTH      | On-die termination signal for DDR2, DDR3, and LPDDR3 memory devices. (Do not confuse this memory device signal with the FPGA's internal on-chip termination signal.) |
| afi_par             | Input     | AFI_CS_WIDTH          | Address and command parity input. (DDR4)<br>Address parity input. (QDR-IV)   |
| afi_ainv            | Input     | AFI_CONTROL_WIDTH     | Address inversion. (QDR-IV)  |
| afi_mem_clk_disable | Input     | AFI_CLK_PAIR_COUNT    | When this signal is asserted, mem_clk and mem_clk_n are disabled. This signal is used in low-power mode.   |
| afi_wps_n           | Output    | AFI_CS_WIDTH          | WPS (for QDR II/II+ memory devices.)   |
| afi_rps_n           | Output    | AFI_CS_WIDTH          | RPS (for QDR II/II+ memory devices.)   |

### 2.17.3.3. AFI Write Data Signals

Write Data Signals for AFI 4.0 control the data, data mask, and strobe signals passed to the memory device during write operations.

**Table 33. Write Data Signals**

| Signal Name   | Direction | Width          | Description  |
|---------------|-----------|----------------|--|
| afi_dqs_burst | Input     | AFI_RATE_RATIO | Controls the enable on the strobe (DQS) pins for DDR2, DDR3, LPDDR2, and LPDDR3 memory devices. When this signal is asserted, mem_dqs and mem_dqsn are driven. |

*continued...*



| Signal Name     | Direction | Width                   | Description  |
|-----------------|-----------|-------------------------|--|
|                 |           |                         | This signal must be asserted before afi_wdata_valid to implement the write preamble, and must be driven for the correct duration to generate a correctly timed mem_dqs signal.   |
| afi_wdata_valid | Input     | AFI_RATE_RATIO          | Write data valid signal. This signal controls the output enable on the data and data mask pins.  |
| afi_wdata       | Input     | AFI_DQ_WIDTH            | Write data signal to send to the memory device at double-data rate. This signal controls the PHY's mem_dq output.  |
| afi_dm          | Input     | AFI_DM_WIDTH            | Data mask. This signal controls the PHY's mem_dm signal for DDR2, DDR3, LPDDR2, LPDDR3, and RLDRAM II memory devices.) Also directly controls the PHY's mem_dbi signal for DDR4. The mem_dm and mem_dbi features share the same port on the memory device. |
| afi_bws_n       | Input     | AFI_DM_WIDTH            | Data mask. This signal controls the PHY's mem_bws_n signal for QDR II/II+ memory devices.  |
| afi_dinv        | Input     | AFI_WRITE_DQS_WIDTH * 2 | Data inversion. It directly controls the PHY's mem_dinva/b signal for QDR-IV devices.  |

#### 2.17.3.4. AFI Read Data Signals

Read Data Signals for AFI 4.0 control the data sent from the memory device during read operations.

**Table 34. Read Data Signals**

| Signal Name       | Direction | Width          | Description  |
|-------------------|-----------|----------------|--|
| afi_rdata_en_full | Input     | AFI_RATE_RATIO | Read data enable full. Indicates that the memory controller is currently performing a read operation. This signal is held high for the entire read burst. If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2). |
| afi_rdata         | Output    | AFI_DQ_WIDTH   | Read data from the memory device. This data is considered valid only when afi_rdata_valid is asserted by the PHY.  |
| afi_rdata_valid   | Output    | AFI_RATE_RATIO | Read data valid. When asserted, this signal indicates that the afi_rdata bus is valid. If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2).  |



### 2.17.3.5. AFI Calibration Status Signals

The PHY instantiates a sequencer which calibrates the memory interface with the memory device and some internal components such as read FIFOs and valid FIFOs. The sequencer reports the results of the calibration process to the controller through the Calibration Status Signals in the AFI interface.

**Table 35. Calibration Status Signals**

| Signal Name  | Direction | Width          | Description   |
|--|-----------|----------------|---|
| afi_cal_success  | Output    | 1              | Asserted to indicate that calibration has completed successfully.   |
| afi_cal_fail   | Output    | 1              | Asserted to indicate that calibration has failed.   |
| afi_cal_req  | Input     | 1              | Effectively a synchronous reset for the sequencer. When this signal is asserted, the sequencer returns to the reset state; when this signal is released, a new calibration sequence begins.   |
| afi_wlat   | Output    | AFI_WLAT_WIDTH | The required write latency in afi_clk cycles, between address/command and write data being issued at the PHY/controller interface. The afi_wlat value can be different for different groups; each group's write latency can range from 0 to 63. If write latency is the same for all groups, only the lowest 6 bits are required. |
| afi_rlat<br><i>(1)</i>   | Output    | AFI_RLAT_WIDTH | The required read latency in afi_clk cycles between address/command and read data being returned to the PHY/controller interface. Values can range from 0 to 63.  |
| Note to Table:   |           |                |   |
| 1. The afi_rlat signal is not supported for PHY-only designs. Instead, you can sample the afi_rdata_valid signal to determine when valid read data is available. |           |                |   |

### 2.17.3.6. AFI Tracking Management Signals

When tracking management is enabled, the sequencer can take control over the AFI 4.0 interface at given intervals, and issue commands to the memory device to track the internal DQS Enable signal alignment to the DQS signal returning from the memory device. The tracking management portion of the AFI 4.0 interface provides a means for the sequencer and the controller to exchange handshake signals.

**Table 36. Tracking Management Signals**

| Signal Name          | Direction | Width | Description   |
|----------------------|-----------|-------|---|
| afi_ctl_refresh_done | Input     | 4     | Handshaking signal from controller to tracking manager, indicating that a refresh has occurred and waiting for a response.                                  |
| afi_seq_busy         | Output    | 4     | Handshaking signal from sequencer to controller, indicating when DQS tracking is in progress.   |
| afi_ctl_long_idle    | Input     | 4     | Handshaking signal from controller to tracking manager, indicating that it has exited low power state without a periodic refresh, and waiting for response. |



### 2.17.3.7. AFI Shadow Register Management Signals

Shadow registers are a feature that enables high-speed multi-rank support. Shadow registers allow the sequencer to calibrate each rank separately, and save the calibrated settings—such as deskew delay-chain configurations—of each rank in its own set of shadow registers.

During a rank-to-rank switch, the correct set of calibrated settings is restored just in time to optimize the data valid window. The PHY relies on additional AFI signals to control which set of shadow registers to activate.

**Table 37. Shadow Register Management Signals**

| Signal Name | Direction | Width           | Description   |
|-------------|-----------|-----------------|---|
| afi_wrrank  | Input     | AFI_WRANK_WIDTH | Signal from controller specifying which rank the write data is going to. The signal timing is identical to that of afi_dqs_burst. That is, afi_wrrank must be asserted at the same time and must last the same duration as the afi_dqs_burst signal.  |
| afi_rrank   | Output    | AFI_RRANK_WIDTH | Signal from controller specifying which rank is being read. The signal must be asserted at the same time as the afi_rdata_en signal when issuing a read command, but unlike afi_rdata_en, afi_rrank is stateful. That is, once asserted, the signal value must remain unchanged until the controller issues a new read command to a different rank. |

Both the afi\_wrrank and afi\_rrank signals encode the rank being accessed using the one-hot scheme (e.g. in a quad-rank interface, 0001, 0010, 0100, 1000 refer to the 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> rank respectively). The ordering within the bus is the same as other AFI signals. Specifically the bus is ordered by time slots, for example:

```
Half-rate afi_w/rrank = {T1, T0}
```

```
Quarter-rate afi_w/rrank = {T3, T2, T1, T0}
```

Where Tx is a number of rank-bit words that one-hot encodes the rank being accessed at the y<sup>th</sup> full-rate cycle.



### Additional Requirements for Stratix 10 Shadow Register Support

To ensure that the hardware has enough time to switch from one shadow register to another, the controller must satisfy the following minimum rank-to-rank-switch delays (tRTRS):

- Two read commands going to different ranks must be separated by a minimum of 3 full-rate cycles (in addition to the burst length delay needed to avoid collision of data bursts).
- Two write commands going to different rank must be separated by a minimum of 4 full-rate cycles (in addition to the burst length delay needed to avoid collision of data bursts).

The Stratix 10 device family supports a maximum of 4 sets of shadow registers, each for an independent set of timings. More than 4 ranks are supported if those ranks have four or fewer sets of independent timing. For example, the rank multiplication mode of an LRDIMM allows more than one physical rank to share a set of timing data as a single logical rank. Therefore Stratix 10 devices can support up to 4 logical ranks, though that means more than 4 physical ranks.

## 2.17.4. AFI 4.0 Timing Diagrams

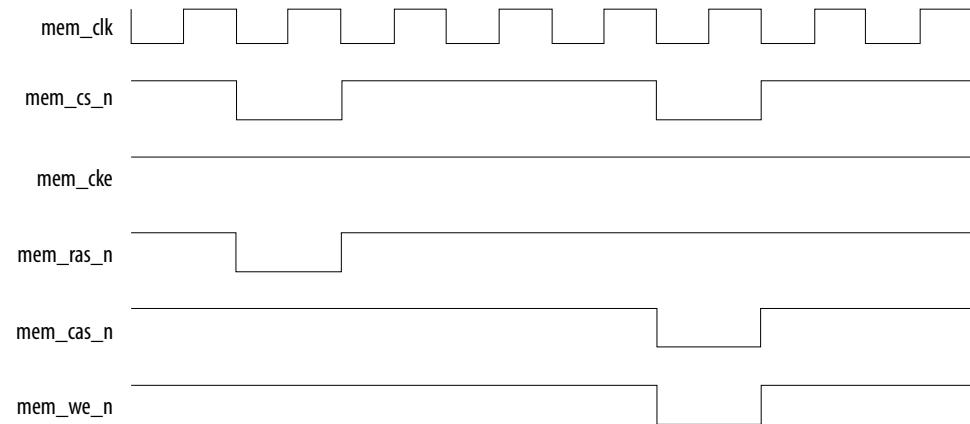
### 2.17.4.1. AFI Address and Command Timing Diagrams

Depending on the ratio between the memory clock and the PHY clock, different numbers of bits must be provided per PHY clock on the AFI interface. The following figures illustrate the AFI address/command waveforms in full, half and quarter rate respectively.

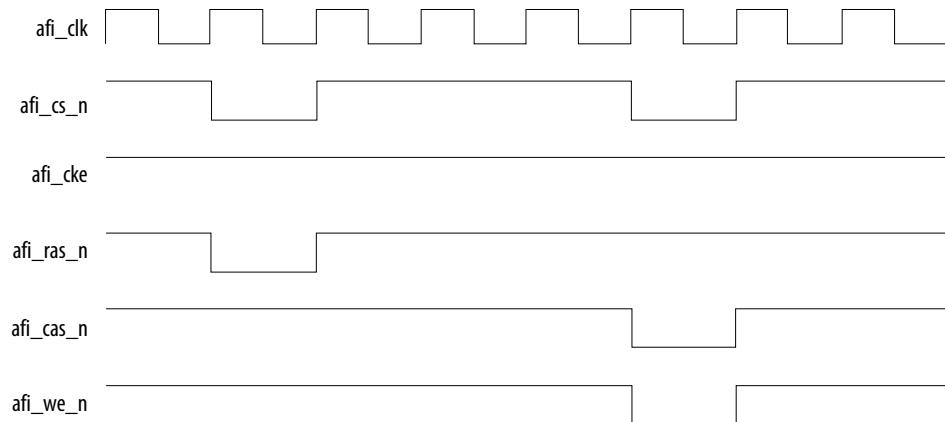
The waveforms show how the AFI command phase corresponds to the memory command output. AFI command 0 corresponds to the first memory command slot, AFI command 1 corresponds to the second memory command slot, and so on.

**Figure 63. AFI Address and Command Full-Rate**

### Memory Interface

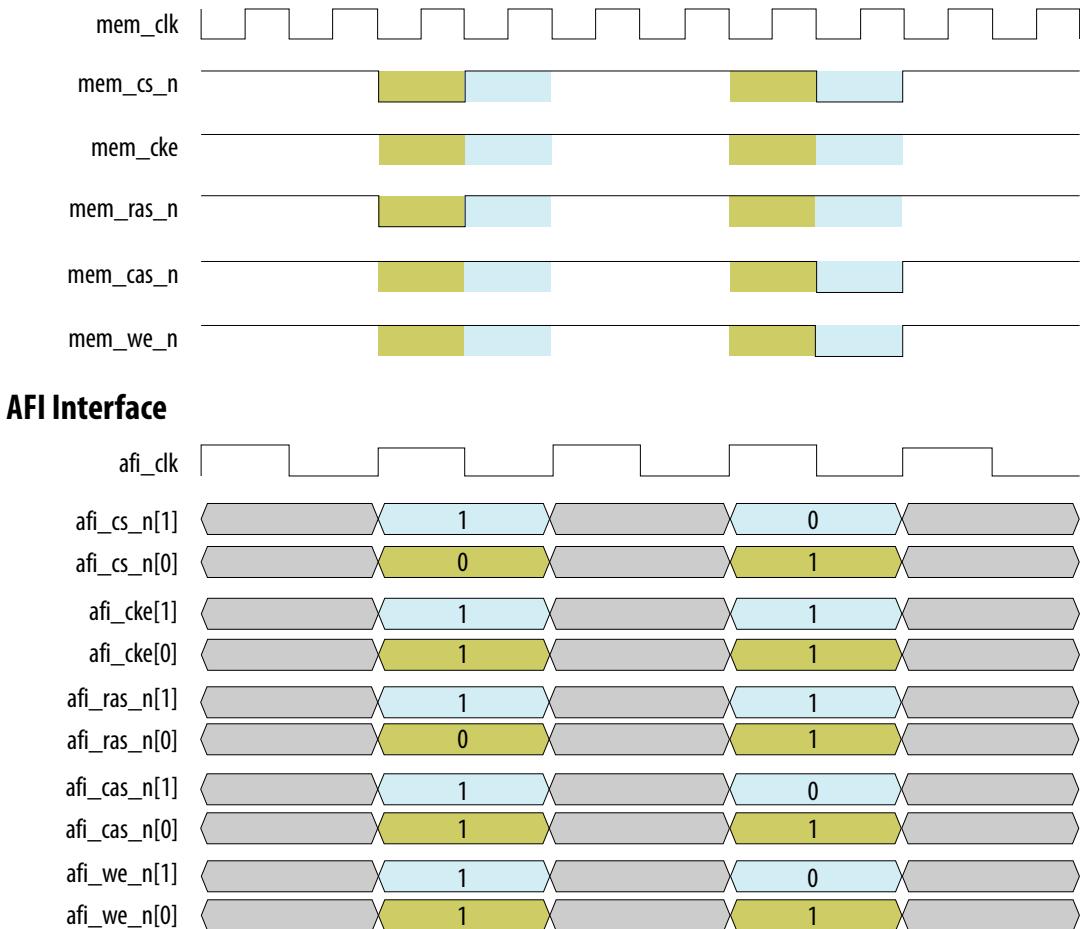


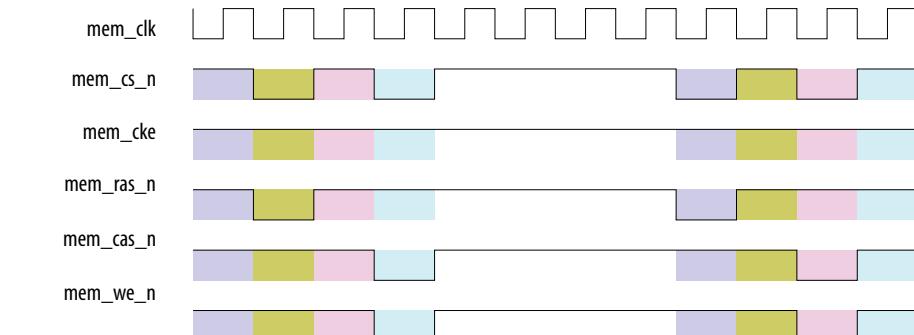
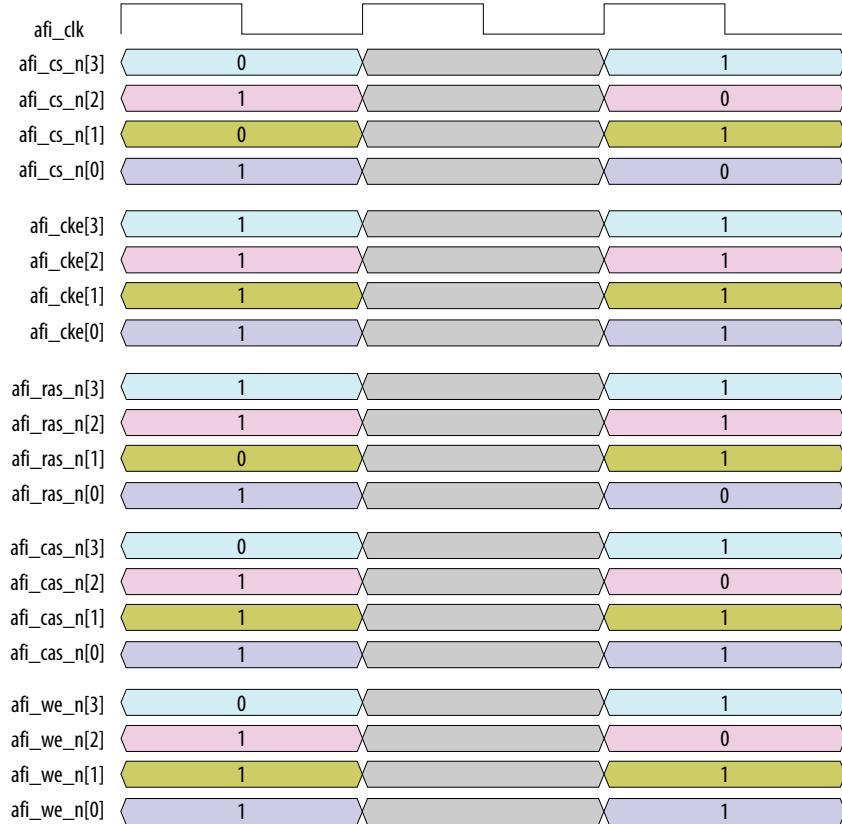
### AFI Interface





**Figure 64. AFI Address and Command Half-Rate Memory Interface**



**Figure 65. AFI Address and Command Quarter-Rate**
**Memory Interface**

**AFI Interface**

**2.17.4.2. AFI Write Sequence Timing Diagrams**

The following timing diagrams illustrate the relationships between the write command and corresponding write data and write enable signals, in full, half, and quarter rate.

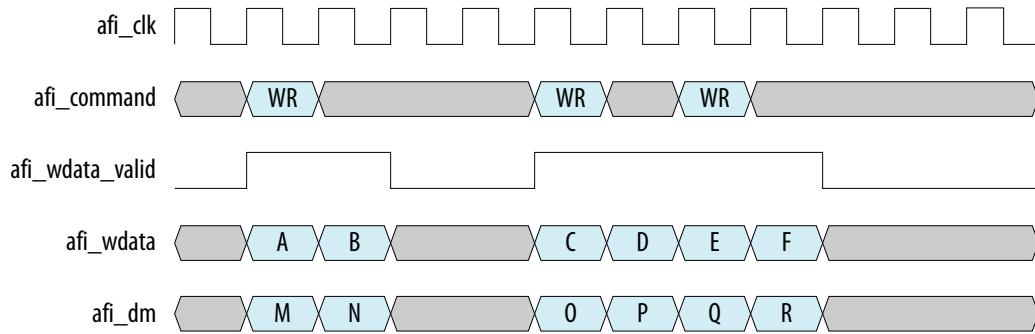
For half rate and quarter rate, when the write command is sent on the first memory clock in a PHY clock (for example,  $\text{afi\_cs\_n}[0] = 0$ ), that access is called *aligned* access; otherwise it is called *unaligned* access. You may use either aligned or unaligned access, or you may use both, but you must ensure that the distance



between the write command and the corresponding write data are constant on the AFI interface. For example, if a command is sent on the second memory clock in a PHY clock, the write data must also start at the second memory clock in a PHY clock.

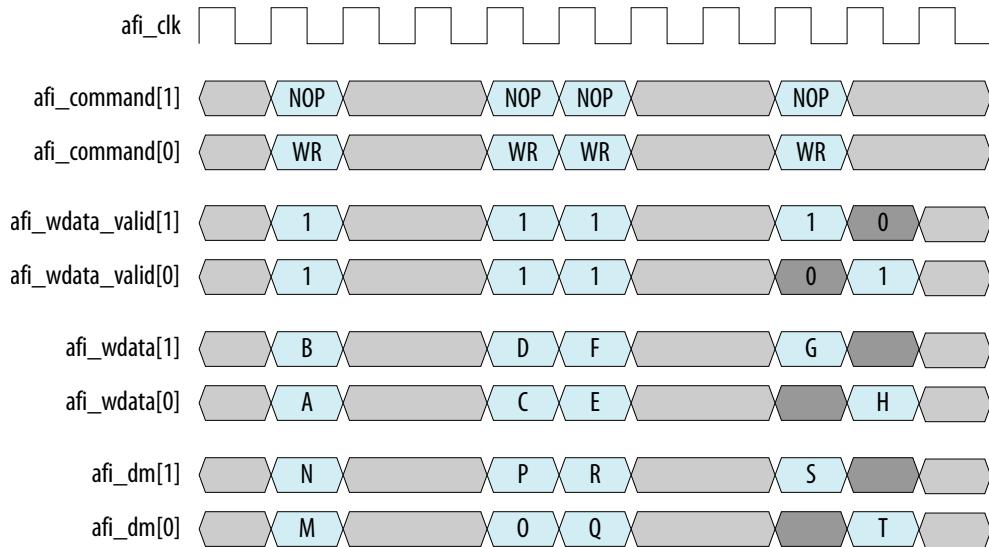
### Write sequences with wlat=0

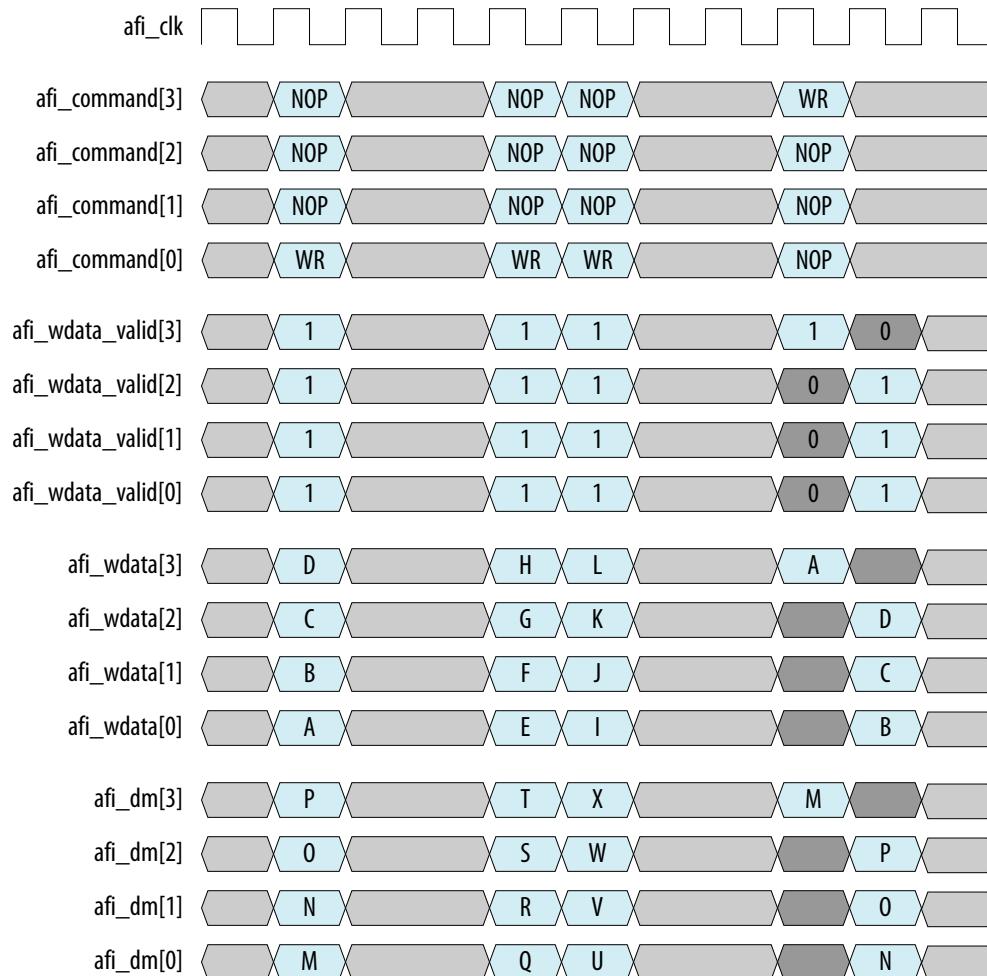
**Figure 66. AFI Write Data Full-Rate, wlat=0**



The following diagrams illustrate both aligned and unaligned access. The first three write commands are aligned accesses where they were issued on LSB of afi\_command. The fourth write command is unaligned access where it was issued on a different command slot. AFI signals must be shifted accordingly, based on the command slot.

**Figure 67. AFI Write Data Half-Rate, wlat=0**



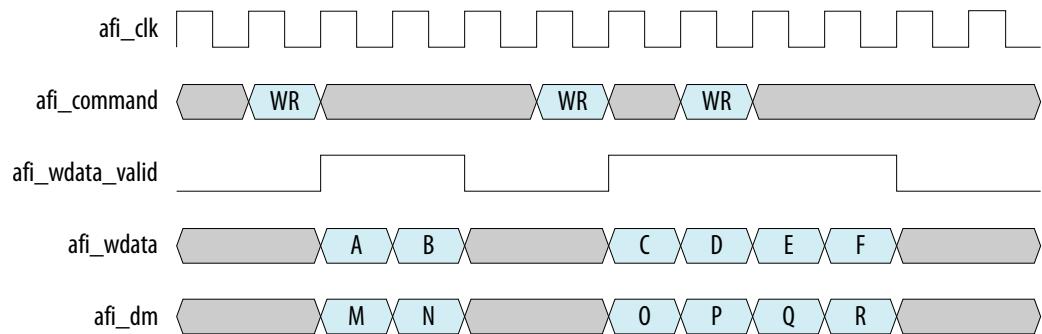
**Figure 68. AFI Write Data Quarter-Rate, wlat=0**


#### Write sequences with wlat=non-zero

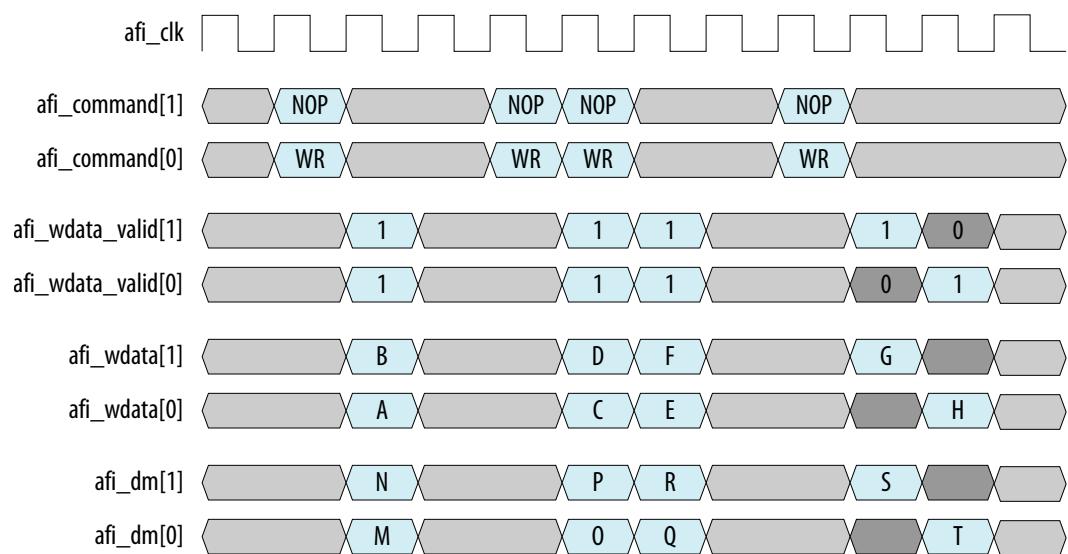
The **afi\_wlat** is a signal from the PHY. The controller must delay **afi\_dqs\_burst**, **afi\_wdata\_valid**, **afi\_wdata** and **afi\_dm** signals by a number of PHY clock cycles equal to **afi\_wlat**, which is a static value determined by calibration before the PHY asserts **cal\_success** to the controller. The following figures illustrate the cases when **wlat=1**. Note that **wlat** is in the number of PHY clocks and therefore **wlat=1** equals 1, 2, and 4 memory clocks delay, respectively, on full, half and quarter rate.

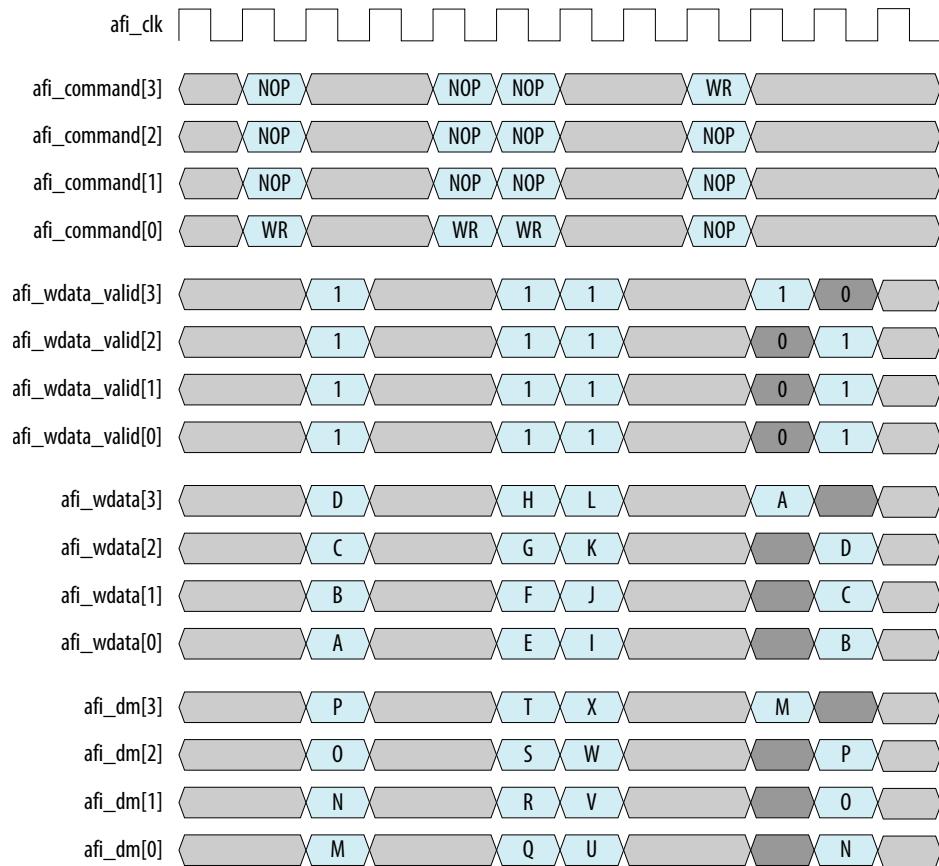


**Figure 69. AFI Write Data Full-Rate, wlat=1**



**Figure 70. AFI Write Data Half-Rate, wlat=1**



**Figure 71. AFI Write Data Quarter-Rate, wlat=1**


### DQS burst

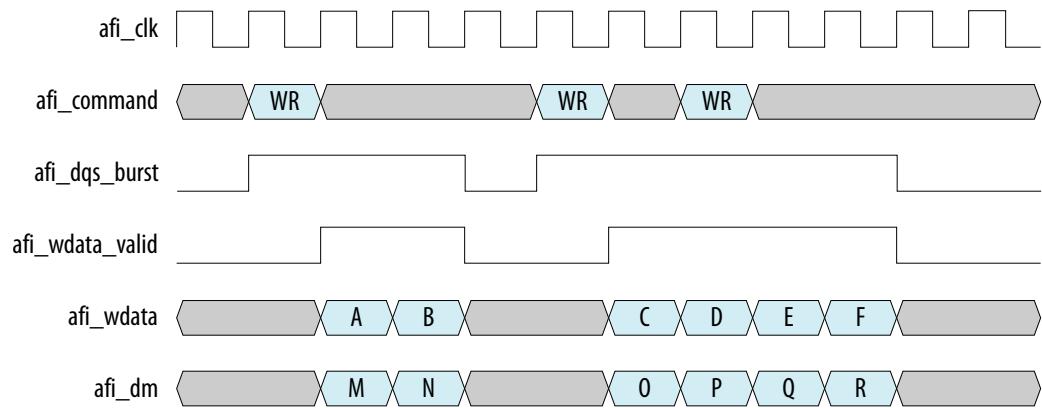
The afi\_dqs\_burst signal must be asserted one or two complete memory clock cycles earlier to generate DQS preamble. DQS preamble is equal to one-half and one-quarter AFI clock cycles in half and quarter rate, respectively.

A DQS preamble of two is required in DDR4, when the write preamble is set to two clock cycles.

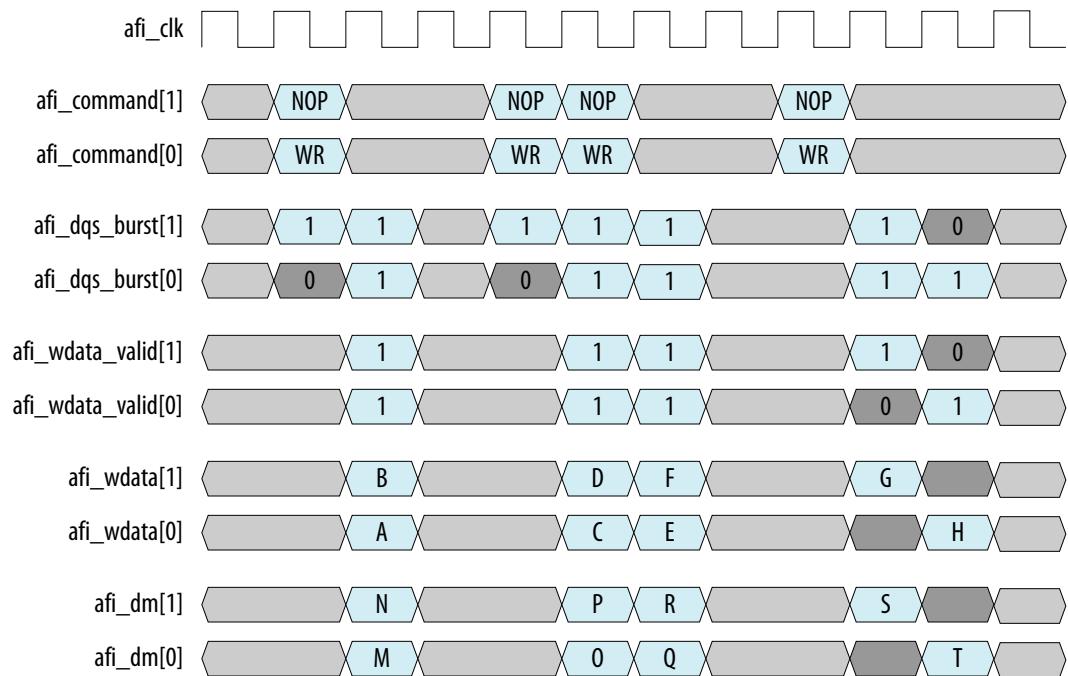
The following diagrams illustrate how afi\_dqs\_burst must be asserted in full, half, and quarter-rate configurations.

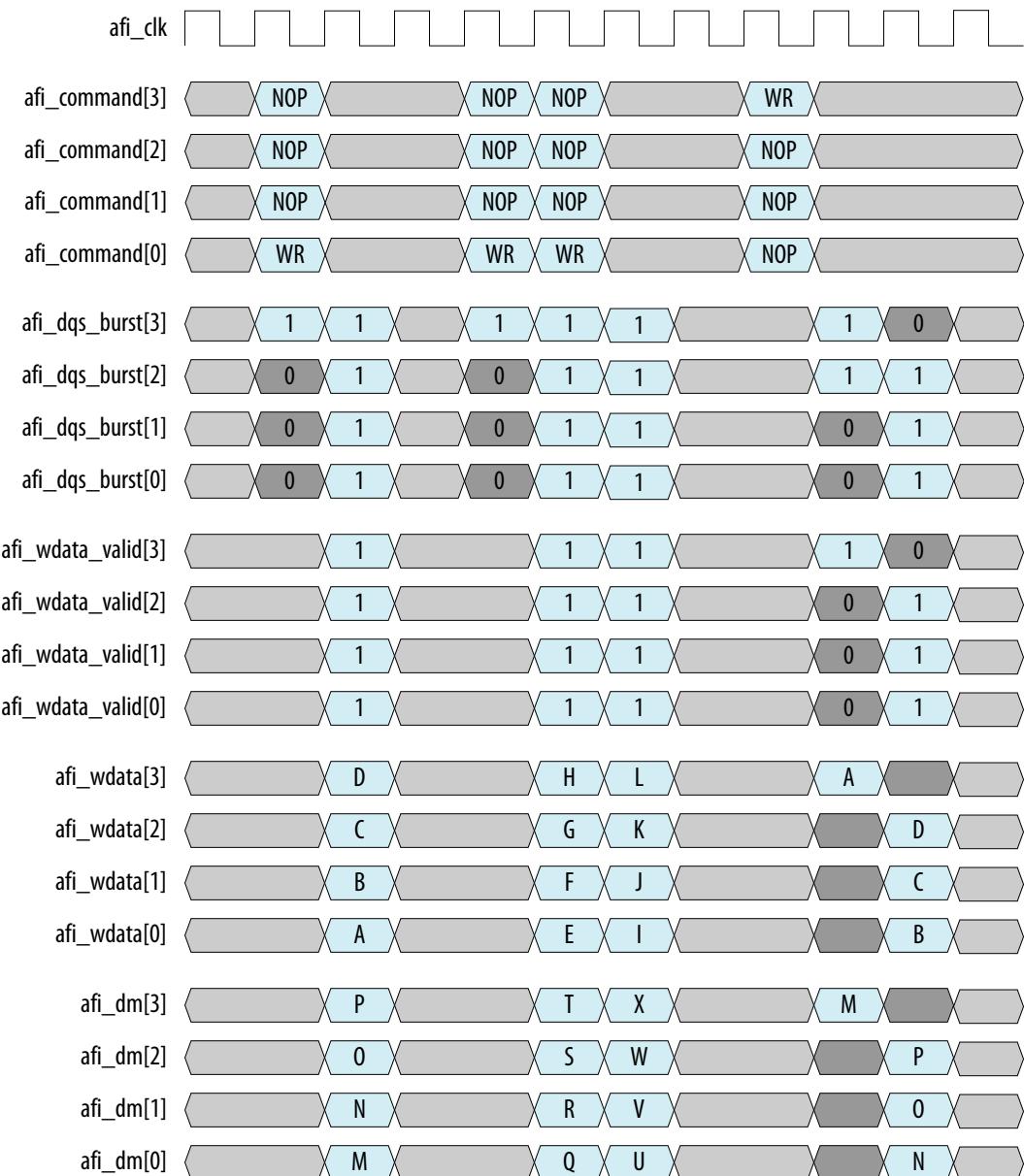


**Figure 72. AFI DQS Burst Full-Rate, wlat=1**



**Figure 73. AFI DQS Burst Half-Rate, wlat=1**



**Figure 74. AFI DQS Burst Quarter-Rate, wlat=1**


#### **Write data sequence with DBI (DDR4 and QDRIV only)**

The DDR4 write DBI feature is supported in the PHY, and when it is enabled, the PHY sends and receives the DBI signal without any controller involvement. The sequence is identical to non-DBI scenarios on the AFI interface.

## Write data sequence with CRC (DDR4 only)

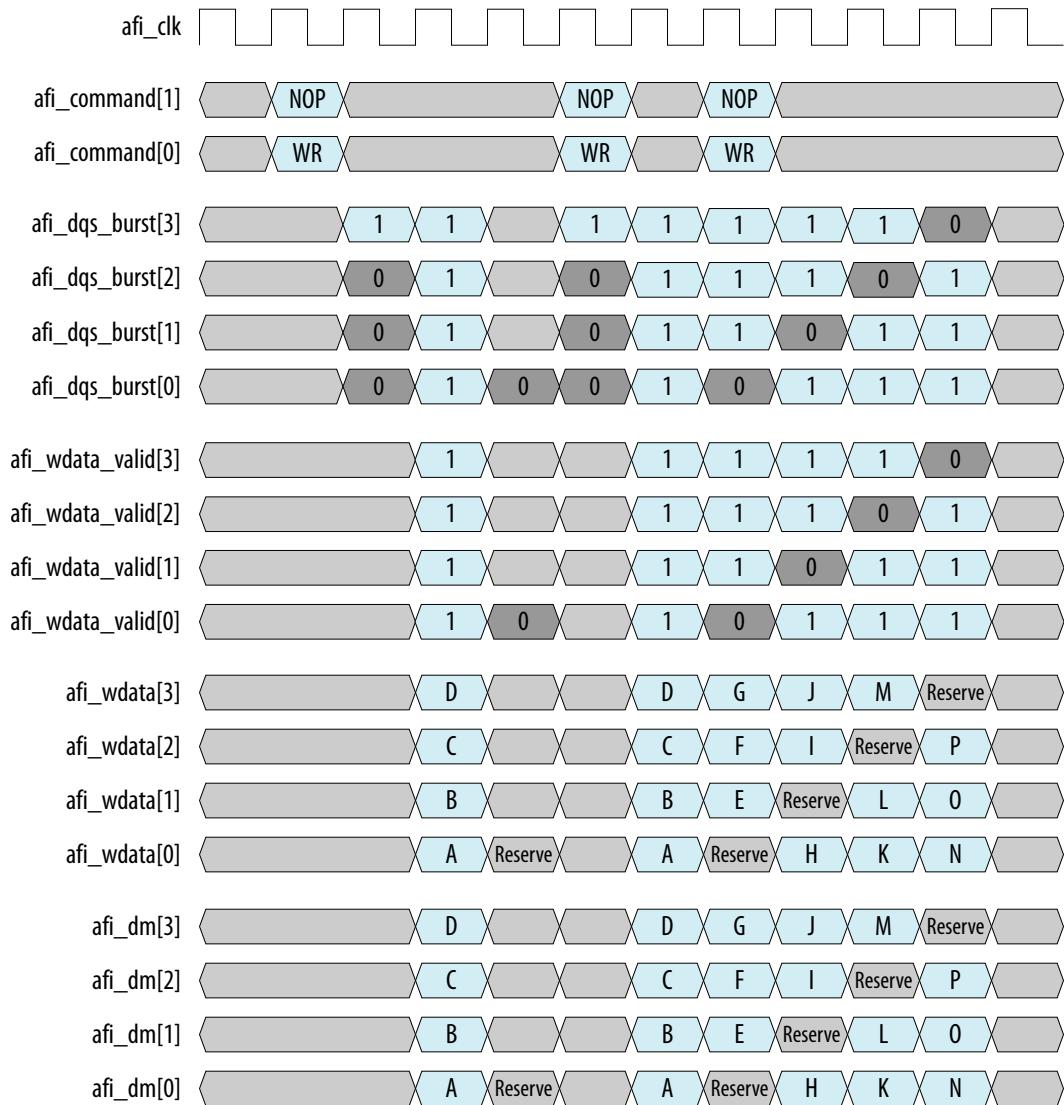
When the CRC feature of the PHY is enabled and used, the controller ensures at least one memory clock cycle between write commands, during which the PHY inserts the CRC data. Sending back to back write command would cause functional failure. The following figures show the legal sequences in CRC mode.

Entries marked as *0* and *RESERVE* must be observed by the controller; no information is allowed on those entries.

**Figure 75.** AFI Write Data with CRC Half-Rate, wlat=2



**Figure 76. AFI Write Data with CRC Quarter-Rate, wlat=2**



#### 2.17.4.3. AFI Read Sequence Timing Diagrams

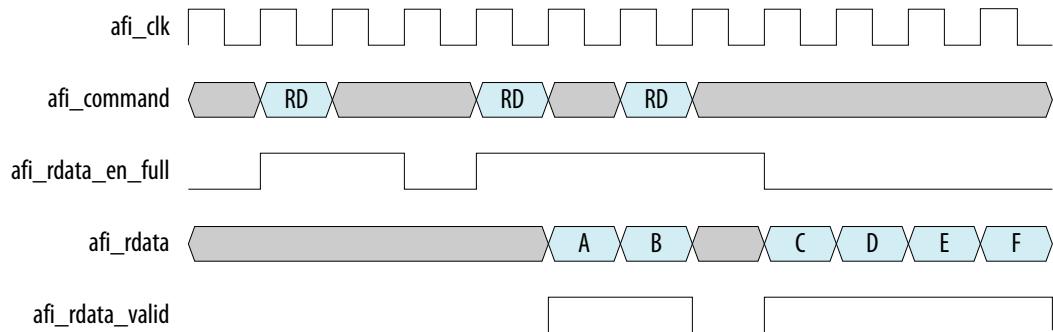
The following waveforms illustrate the AFI write data waveform in full, half, and quarter-rate, respectively.

The `afi_rdata_en_full` signal must be asserted for the entire read burst operation. The `afi_rdata_en` signal need only be asserted for the intended read data.

Aligned and unaligned access for read commands is similar to write commands; however, the `afi_rdata_en_full` signal must be sent on the same memory clock in a PHY clock as the read command. That is, if a read command is sent on the second memory clock in a PHY clock, `afi_rdata_en_full` must also be asserted, starting from the second memory clock in a PHY clock.

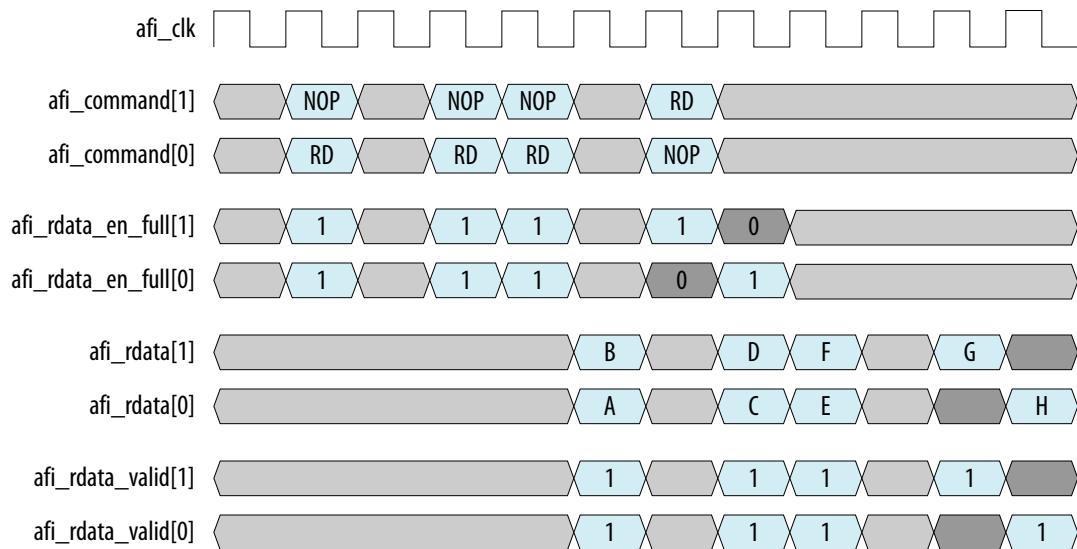


**Figure 77. AFI Read Data Full-Rate**

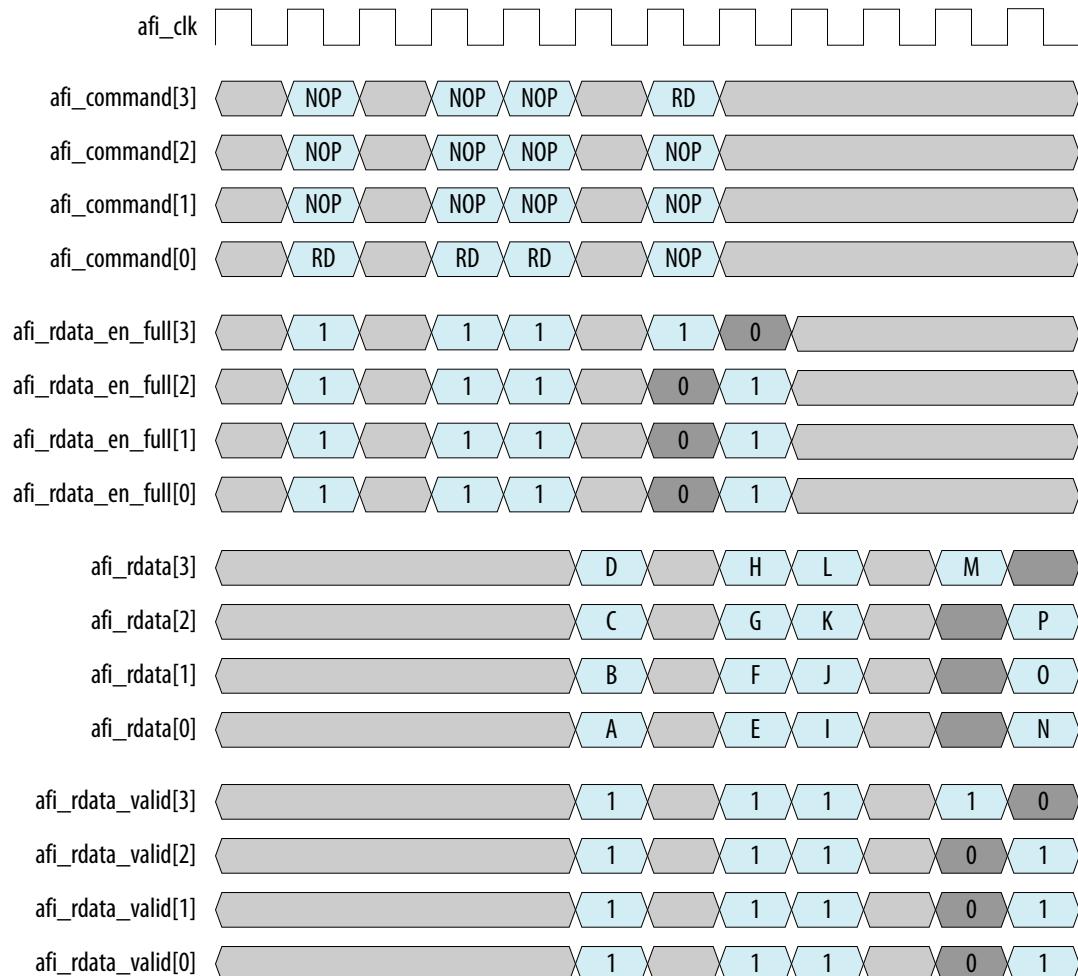


The following figure illustrates that the second and third reads require only the first and second half of data, respectively. The first three read commands are aligned accesses where they are issued on the LSB of afi\_command. The fourth read command is unaligned access, where it is issued on a different command slot. AFI signals must be shifted accordingly, based on command slot.

**Figure 78. AFI Read Data Half-Rate**



In the following figure, the first three read commands are aligned accesses where they are issued on the LSB of afi\_command. The fourth read command is unaligned access, where it is issued on a different command slot. AFI signals must be shifted accordingly, based on command slot.

**Figure 79. AFI Read Data Quarter-Rate**


#### 2.17.4.4. AFI Calibration Status Timing Diagram

The controller interacts with the PHY during calibration at power-up and at recalibration.

At power-up, the PHY holds `afi_cal_success` and `afi_cal_fail` 0 until calibration is done, when it asserts `afi_cal_success`, indicating to controller that the PHY is ready to use and `afi_wlat` and `afi_rlat` signals have valid values.

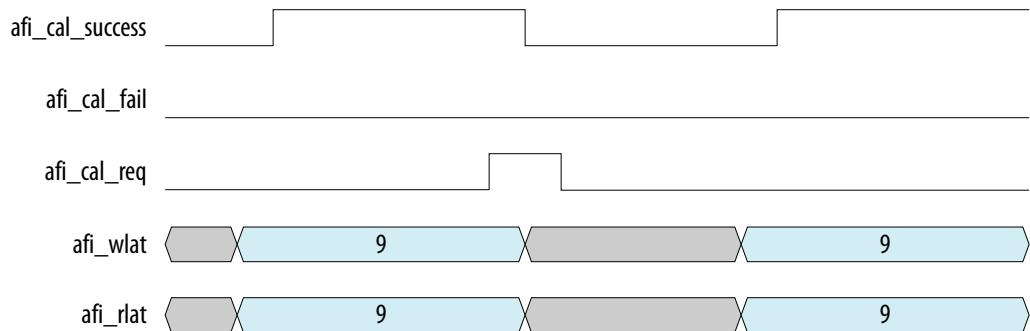
At recalibration, the controller asserts `afi_cal_req`, which triggers the same sequence as at power-up, and forces recalibration of the PHY.



**Figure 80. Calibration**



### AFI Interface



## 2.18. Stratix 10 Resource Utilization

The following tables provide resource utilization information for external memory interfaces on Stratix 10 devices.

### 2.18.1. QDR-IV Resource Utilization in Stratix 10 Devices

The following table shows typical resource usage of QDR-IV controllers for Stratix 10 devices.

**Table 38. QDR-IV Resource Utilization in Stratix 10 Devices**

| Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Block Memory Bits | M20Ks | Soft Controller |
|---------------------|---------------------|---------------------------|-------------------|-------|-----------------|
| 18                  | 2123                | 4592                      | 18432             | 8     | 1               |
| 36                  | 2127                | 6023                      | 36864             | 16    | 1               |
| 72                  | 2114                | 8826                      | 73728             | 32    | 1               |

## 2.19. Stratix 10 EMIF Latency

The following latency data applies to all memory protocols supported by the Stratix 10 EMIF IP.

**Table 39. Latency in Full-Rate Memory Clock Cycles**

| Rate <sup>1</sup> | Controller Address & Command | PHY Address & Command | Memory Read Latency <sup>2</sup> | PHY Read Data Return | Controller Read Data Return | Round Trip | Round Trip Without Memory |
|-------------------|------------------------------|-----------------------|----------------------------------|----------------------|-----------------------------|------------|---------------------------|
| Half:Write        | 12                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Half:Read         | 8                            | 2                     | 3-23                             | 6                    | 8                           | 27-47      | 24                        |

*continued...*



| Rate <sup>1</sup>   | Controller Address & Command | PHY Address & Command | Memory Read Latency <sup>2</sup> | PHY Read Data Return | Controller Read Data Return | Round Trip | Round Trip Without Memory |
|---------------------|------------------------------|-----------------------|----------------------------------|----------------------|-----------------------------|------------|---------------------------|
| Quarter:Write       | 14                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Quarter:Read        | 10                           | 2                     | 3-23                             | 6                    | 14                          | 35-55      | 32                        |
| Half:Write (ECC)    | 14                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Half:Read (ECC)     | 12                           | 2                     | 3-23                             | 6                    | 8                           | 31-51      | 28                        |
| Quarter:Write (ECC) | 14                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Quarter:Read (ECC)  | 12                           | 2                     | 3-23                             | 6                    | 14                          | 37-57      | 34                        |

1. User interface rate; the controller always operates in half rate.  
2. Minimum and maximum read latency range for DDR3, DDR4, and LPDDR3.

## 2.20. Integrating a Custom Controller with the Hard PHY

If you want to use your own custom memory controller, you must integrate the controller with the hard PHY to achieve a complete memory solution.

Observe the following general guidelines:

- When you configure your external memory interface IP, ensure that you select **Configuration > Hard PHY Only** on the **General** tab in the parameter editor.
- Consult the AFI 4.0 Specification, for detailed information on the AFI interface to the PHY.

### Related Information

[AFI 4.0 Specification](#) on page 117

## 2.21. Document Revision History

| Date     | Version    | Changes          |
|----------|------------|------------------|
| May 2017 | 2017.05.08 | Initial release. |



## 3. Functional Description—Intel Arria 10 EMIF IP

Intel Arria 10 devices can interface with external memory devices clocking at frequencies of up to 1.3 GHz. The external memory interface IP component for Arria 10 devices provides a single parameter editor for creating external memory interfaces, regardless of memory protocol. Unlike earlier EMIF solutions which used protocol-specific parameter editors to create memory interfaces via a complex RTL generation method, the Arria 10 EMIF solution captures the protocol-specific hardened EMIF logic of the Arria 10 device together with more generic soft logic.

The Arria 10 EMIF solution is designed with the following implementations in mind:

### **Hard Memory Controller and Hard PHY**

This implementation provides a complete external memory interface, based on the hard memory controller and hard PHY that are part of the Arria 10 silicon. An Avalon-MM interface is available for integration with user logic.

### **Soft Memory Controller and Hard PHY**

This implementation provides a complete external memory interface, using an Intel-provided soft-logic-based memory controller and the hard PHY that is part of the Arria 10 silicon. An Avalon-MM interface is available for integration with user logic.

### **Custom Memory Controller and Hard PHY (PHY only)**

This implementation provides access to the AFI interface, to allow use of a custom or third-party memory controller with the hard PHY that is part of the Arria 10 silicon. Because only the PHY component is provided by Intel, this configuration is also known as *PHY only*.

### **Related Information**

- [Arria 10 EMIF Architecture: Introduction](#) on page 143
- [Hardware Resource Sharing Among Multiple EMIFs](#) on page 154
- [Arria 10 EMIF IP Component](#) on page 158
- [Compiling Arria 10 EMIF IP with the Quartus Prime Software](#) on page 180
- [Debugging Arria 10 EMIF IP](#) on page 181
- [Intel Arria 10 Core Fabric and General Purpose I/Os Handbook](#)

### 3.1. Supported Memory Protocols

The following table lists the external memory protocols supported by Arria 10 devices.

---

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Empirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.

ISO  
9001:2015  
Registered

**Table 40. Supported Memory Protocols**

| Protocol              | Hard Controller and Hard PHY | Soft Controller and Hard PHY | PHY Only |
|-----------------------|------------------------------|------------------------------|----------|
| DDR4                  | Yes                          | —                            | Yes      |
| DDR3                  | Yes                          | —                            | Yes      |
| LPDDR3                | Yes                          | —                            | Yes      |
| RDRAM 3               | —                            | Third party                  | Yes      |
| QDR II/II+/II+ Xtreme | —                            | Yes                          | —        |
| QDR-IV                | —                            | Yes                          | —        |

Memory protocols not listed above are not supported by the Arria 10 EMIF IP; however, you can implement a custom memory interface for these protocols using the Altera PHYLITE Megafunction.

LPDDR3 is supported for simulation, compilation, and timing. Hardware support for LPDDR3 will be provided in a future release.

**Note:** To achieve maximum top-line spec performance for a DDR4 interface, both read data bus inversion and periodic OCT calibration must be enabled. For maximum top-line spec performance for a QDR-IV interface, read data bus inversion must be enabled.

#### Related Information

[Altera PHYLITE for Memory Megafunction User Guide](#)

## 3.2. Key Differences Compared to UniPHY IP and Previous Device Families

The Arria 10 EMIF IP has a new design which bears several notable differences compared to UniPHY-based IP. If you are familiar with the UniPHY-based IP, you should review the following differences, as they affect the way you generate, instantiate, and use the Arria 10 EMIF IP.

- Unlike the UniPHY-based IP, which presents a protocol-specific parameter editor for each supported memory protocol, the Arria 10 EMIF IP uses one parameter editor for all memory protocols.
- With UniPHY-based IP, you must run the `<variation_name>.pin_assignments.tcl` script following synthesis, to apply I/O assignments to the project's `.qsf` file. In Arria 10 EMIF IP, the `<variation_name>.pin_assignments.tcl` script is no longer necessary. All the I/O assignments are included in the generated `.qip` file, which the Quartus Prime software processes during compilation. Assignments that you make in the `.qsf` file override those in the `.qip` file.
- The Arria 10 EMIF IP includes a `<variation_name>.readme.txt` file, located in the `/altera_emif_arch_nf_<version>` directory. This file contains important information about the implementation of the IP, including pin location guidelines, information on resource sharing, and signal descriptions.
- To generate the synthesis example design or the simulation example design, you need to run additional scripts after generation.



### 3.3. Migrating from Previous Device Families

There is no automatic migration mechanism for external memory interface IP generated for previous device families.

To migrate an existing EMIF IP from an earlier device family to Arria 10, you must reparameterize and regenerate the IP targeting Arria 10, using either the IP Catalog or Qsys. If you attempt to recompile an existing IP generated for a previous device family, you will encounter errors in the Quartus Prime software.

UniPHY-based IP continues to be supported for previous device families.

### 3.4. Arria 10 EMIF Architecture: Introduction

The Arria 10 EMIF architecture contains many new hardware features designed to meet the high-speed requirements of emerging memory protocols, while consuming the smallest amount of core logic area and power.

The following are key hardware features of the Arria 10 EMIF architecture:

#### Hard Sequencer

The sequencer employs a hard Nios II processor, and can perform memory calibration for a wide range of protocols. You can share the sequencer among multiple memory interfaces of the same or different protocols.

#### Hard PHY

The hard PHY in Arria 10 devices can interface with external memories running at speeds of up to 1.3 GHz. The PHY circuitry is hardened in the silicon, which simplifies the challenges of achieving timing closure and minimal power consumption.

#### Hard Memory Controller

The hard memory controller reduces latency and minimizes core logic consumption in the external memory interface. The hard memory controller supports the DDR3, DDR4, and LPDDR3 memory protocols.

#### PHY-Only Mode

Protocols that use a hard controller (DDR4, DDR3, and LPDDR3) as well as RLDRAM 3, provide a "PHY-only" option. When selected, this option generates only the PHY and sequencer, but not the controller. This PHY-Only mode provides a mechanism by which to integrate your own custom soft controller.

#### High-Speed PHY Clock Tree

Dedicated high speed PHY clock networks clock the I/O buffers in Arria 10 EMIF IP. The PHY clock trees exhibit low jitter and low duty cycle distortion, maximizing the data valid window.

#### Automatic Clock Phase Alignment

Automatic clock phase alignment circuitry dynamically adjust the clock phase of core clock networks to match the clock phase of the PHY clock networks. The clock phase alignment circuitry minimizes clock skew that can complicate timing closure in transfers between the FPGA core and the periphery.

## Resource Sharing

The Arria 10 architecture simplifies resource sharing between memory interfaces. Resources such as the OCT calibration block, PLL reference clock pin, and core clock can be shared. The hard Nios processor in the I/O AUX must be shared across all interfaces in a column.

## Related Information

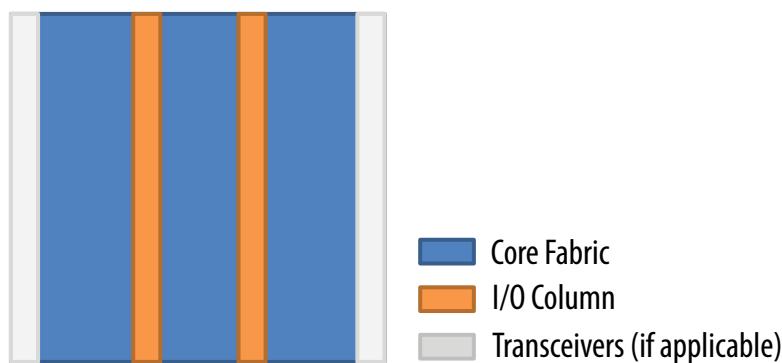
[Intel Arria 10 Core Fabric and General Purpose I/Os Handbook](#)

### 3.4.1. Arria 10 EMIF Architecture: I/O Subsystem

The I/O subsystem consists of two columns inside the core of Arria 10 devices.

Each column can be thought of as loosely analogous to an I/O bank.

**Figure 81. Arria 10 I/O Subsystem**



The I/O subsystem provides the following features:

- General-purpose I/O registers and I/O buffers
- On-chip termination control (OCT)
- I/O PLLs for external memory interfaces and user logic
- Low-voltage differential signaling (LVDS)
- External memory interface components, as follows:
  - Hard memory controller
  - Hard PHY
  - Hard Nios processor and calibration logic
  - DLL

## Related Information

[Arria 10 Core Fabric and General Purpose I/Os Handbook](#)

### 3.4.2. Arria 10 EMIF Architecture: I/O Column

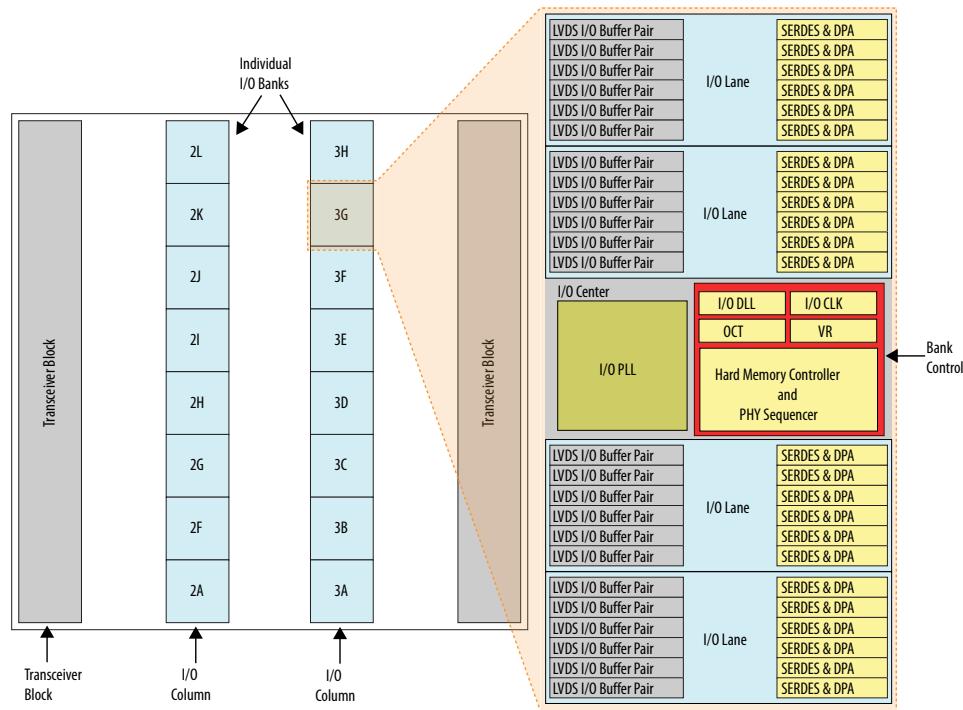
Arria 10 devices have two I/O columns, which contain the hardware related to external memory interfaces.

Each I/O column contains the following major parts:



- A hardened Nios processor with dedicated memory. This Nios block is referred to as the I/O AUX.
- Up to 13 I/O banks. Each I/O bank contains the hardware necessary for an external memory interface.

**Figure 82. I/O Column**



### 3.4.3. Arria 10 EMIF Architecture: I/O AUX

Each column includes one I/O AUX, which contains a hardened Nios II processor with dedicated memory. The I/O AUX is responsible for calibration of all the EMIFs in the column.

The I/O AUX includes dedicated memory which stores both the calibration algorithm and calibration run-time data. The hardened Nios II processor and the dedicated memory can be used only by an external memory interface, and cannot be employed for any other use. The I/O AUX can interface with soft logic, such as the debug toolkit, via an Avalon-MM bus.

The I/O AUX is clocked by an on-die oscillator, and therefore does not consume a PLL.

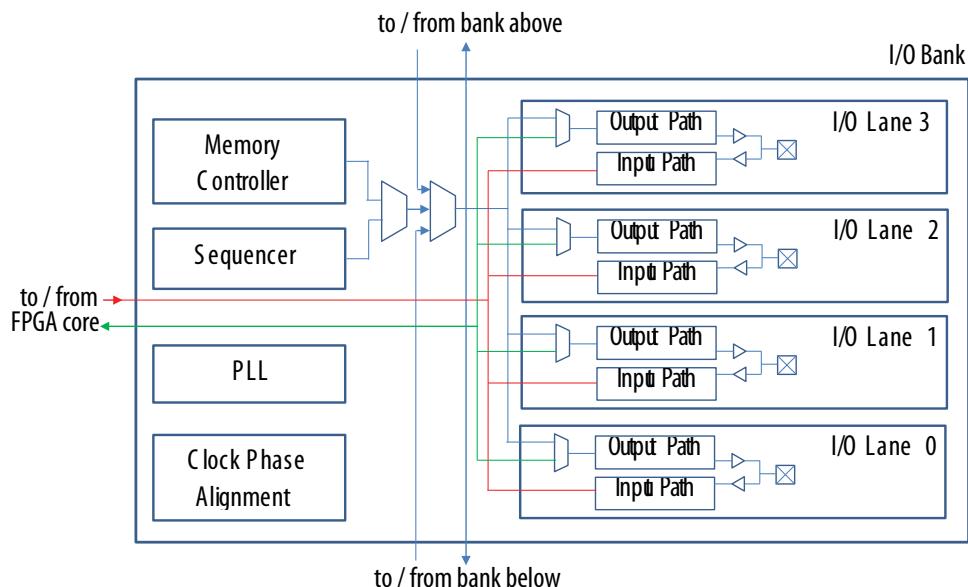
### 3.4.4. Arria 10 EMIF Architecture: I/O Bank

A single I/O bank contains all the hardware needed to build an external memory interface. Each I/O column contains up to 13 I/O banks; the exact number of banks depends on device size and pin package. You can make a wider interface by connecting multiple banks together.

Each I/O bank resides in an I/O column, and contains the following components:

- Hard memory controller
- Sequencer components
- PLL and PHY clock trees
- DLL
- Input DQS clock trees
- 48 pins, organized into four I/O lanes of 12 pins each

**Figure 83. I/O Bank Architecture in Arria 10 Devices**



### I/O Bank Usage

The pins in an I/O bank can serve as address and command pins, data pins, or clock and strobe pins for an external memory interface. You can implement a narrow interface, such as a DDR3 or DDR4 x8 interface, with only a single I/O bank. A wider interface, such as x72 or x144, can be implemented by configuring multiple adjacent banks in a multi-bank interface. Any pins in a bank which are not used by the external memory interface remain available for use as general purpose I/O pins (of the same voltage standard).

Every I/O bank includes a hard memory controller which you can configure for DDR3 or DDR4. In a multi-bank interface, only the controller of one bank is active; controllers in the remaining banks are turned off to conserve power.



To use a multi-bank Arria 10 EMIF interface, you must observe the following rules:

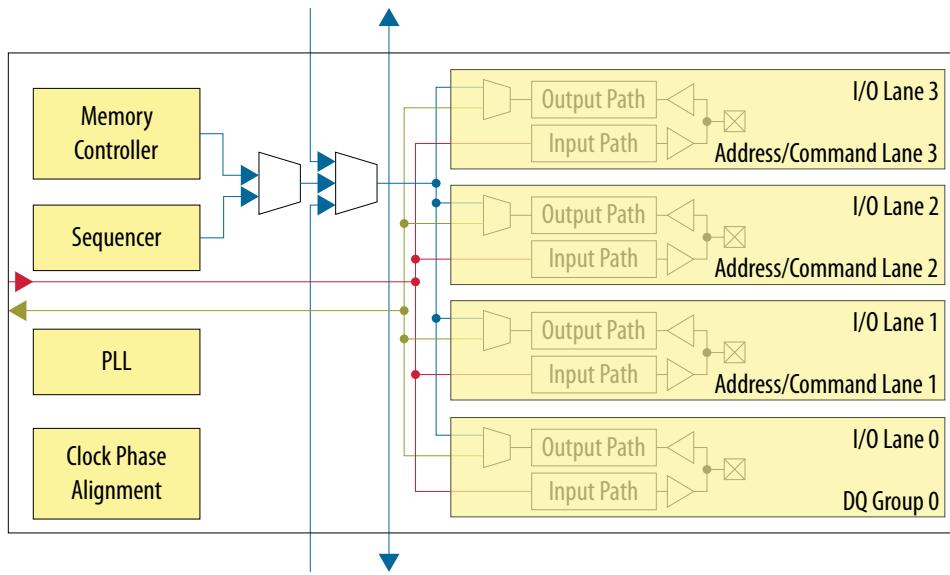
- Designate one bank as the address and command bank.
- The address and command bank must contain all the address and command pins.
- The locations of individual address and command pins within the address and command bank must adhere to the pin map defined in the pin table— regardless of whether you use the hard memory controller or not.
- If you do use the hard memory controller, the address and command bank contains the active hard controller.

All the I/O banks in a column are capable of functioning as the address and command bank. However, for minimal latency, you should select the center-most bank of the interface as the address and command bank.

#### 3.4.4.1. Implementing a x8 Interface with Hard Memory Controller

The following diagram illustrates the use of a single I/O bank to implement a DDR3 or DDR4 x8 interface using the hard memory controller.

**Figure 84. Single Bank x8 Interface With Hard Controller**



In the above diagram, shaded cells indicate resources that are in use.

**Note:**

For information on the I/O lanes and pins in use, consult the pin table for your device or the `<variation_name>/altera_emif_arch_nf_140/<synth/sim>/<variation_name>_altera_emif_arch_nf_140_<unique ID>_readme.txt` file generated with your IP.

#### Related Information

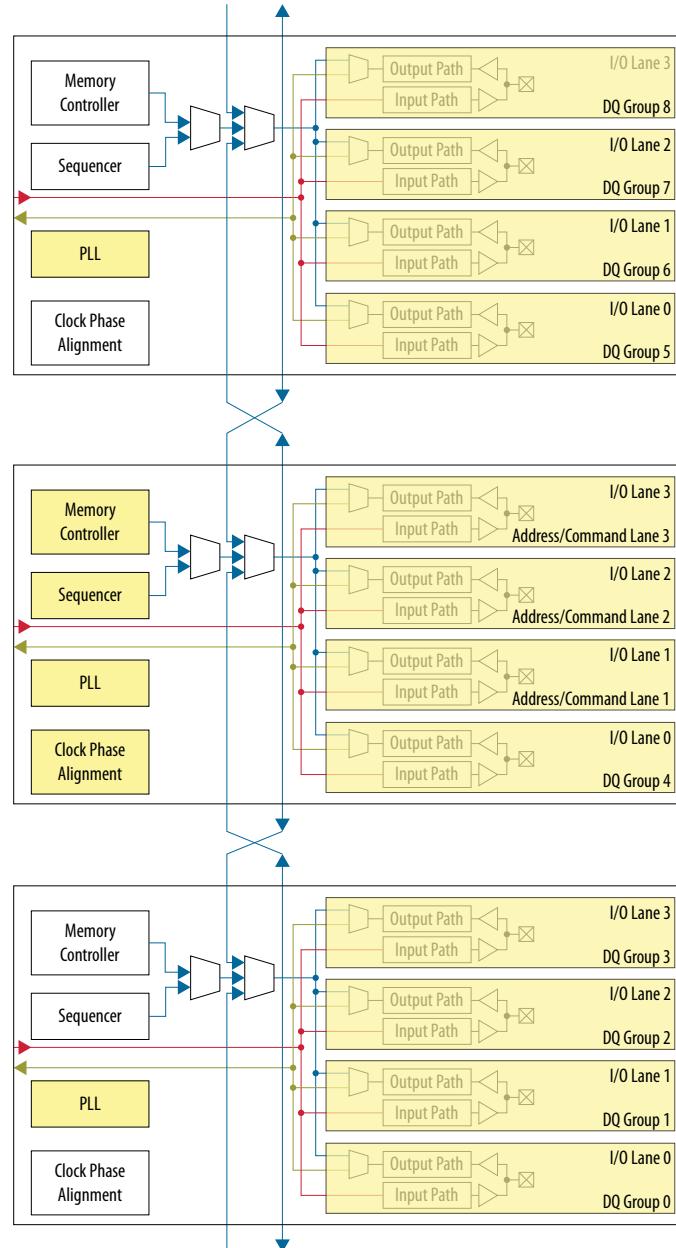
[Intel Arria 10 Core Fabric and General Purpose I/Os Handbook](#)

### 3.4.4.2. Implementing a x72 Interface with Hard Memory Controller

The following diagram illustrates one possible implementation of a DDR3 or DDR4 x72 interface using the hard memory controller.

Note that only the hard memory controller in the address and command bank is used. Similarly, only the clock phase alignment block of the address and command bank is used to generate clock signals for the FPGA core.

**Figure 85. Multi-Bank x72 Interface With Hard Controller**



In the above diagram, shaded cells indicate resources that are in use.



**Note:** For information on the I/O lanes and pins in use, consult the pin table for your device or the <variation\_name>/altera\_emif\_arch\_nf\_140/<synth/sim>/<variation\_name>\_altera\_emif\_arch\_nf\_140\_<unique ID>\_readme.txt file generated with your IP.

#### Related Information

[Arria 10 Core Fabric and General Purpose I/Os Handbook](#)

### 3.4.5. Arria 10 EMIF Architecture: I/O Lane

An I/O bank contains 48 I/O pins, organized into four I/O lanes of 12 pins each.

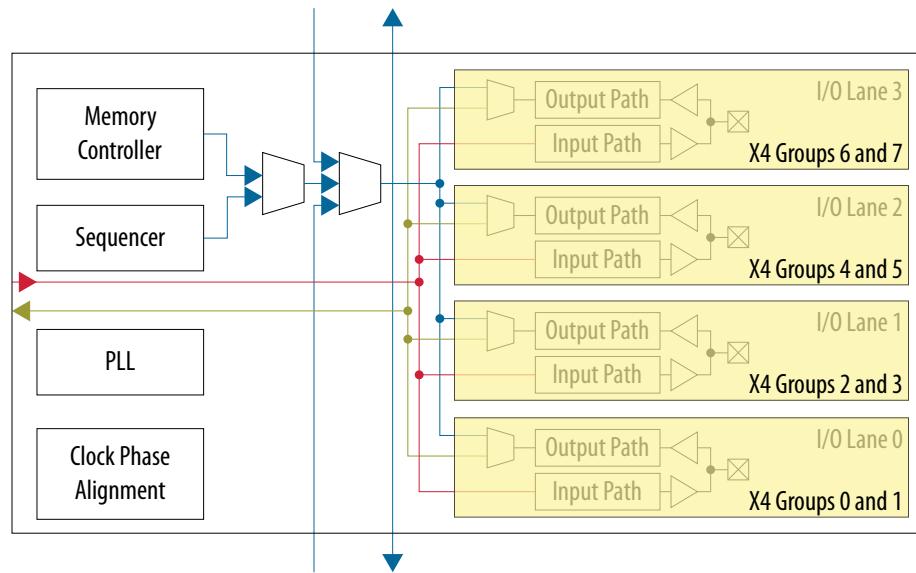
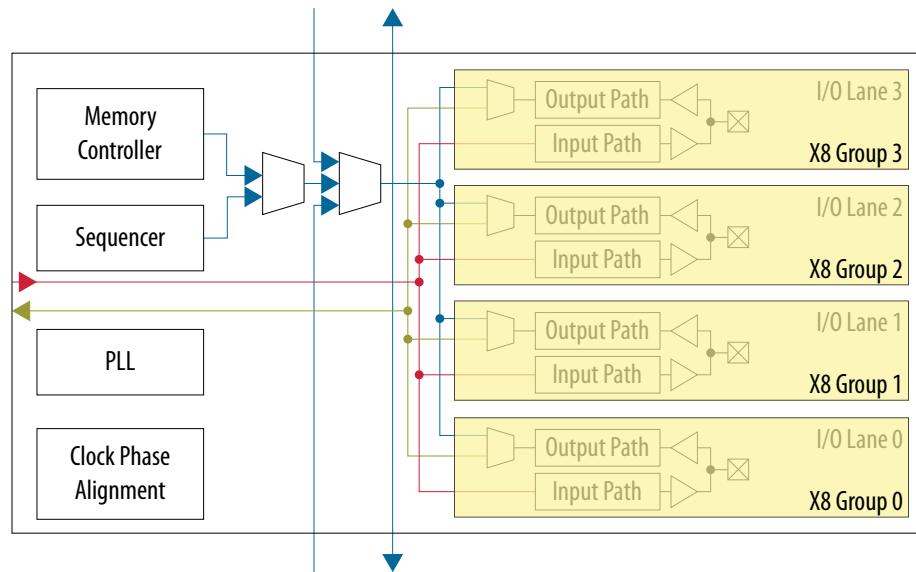
Each I/O lane can implement one x8/x9 read capture group (DQS group), with two pins functioning as the read capture clock/strobe pair (DQS/DQS#), and up to 10 pins functioning as data pins (DQ and DM pins). To implement x18 and x36 groups, you can use multiple lanes within the same bank.

It is also possible to implement a pair of x4 groups in a lane. In this case, four pins function as clock/strobe pair, and 8 pins function as data pins. DM is not available for x4 groups. There must be an even number of x4 groups for each interface.

For x4 groups, DQS0 and DQS1 must be placed in the same I/O lane as a pair. Similarly, DQS2 and DQS3 must be paired. In general, DQS(x) and DQS(x+1) must be paired in the same I/O lane.

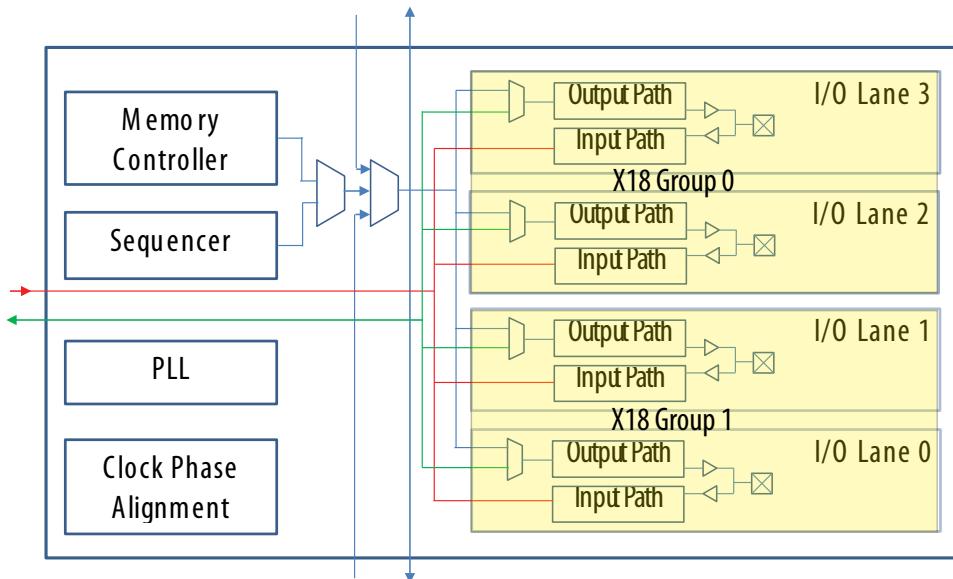
**Table 41. Lanes Used Per Group**

| Group Size | Number of Lanes Used | Maximum Number of Data Pins per Group |
|------------|----------------------|---------------------------------------|
| x8 / x9    | 1                    | 10                                    |
| x18        | 2                    | 22                                    |
| x36        | 4                    | 46                                    |
| pair of x4 | 1                    | 4 per group, 8 per lane               |

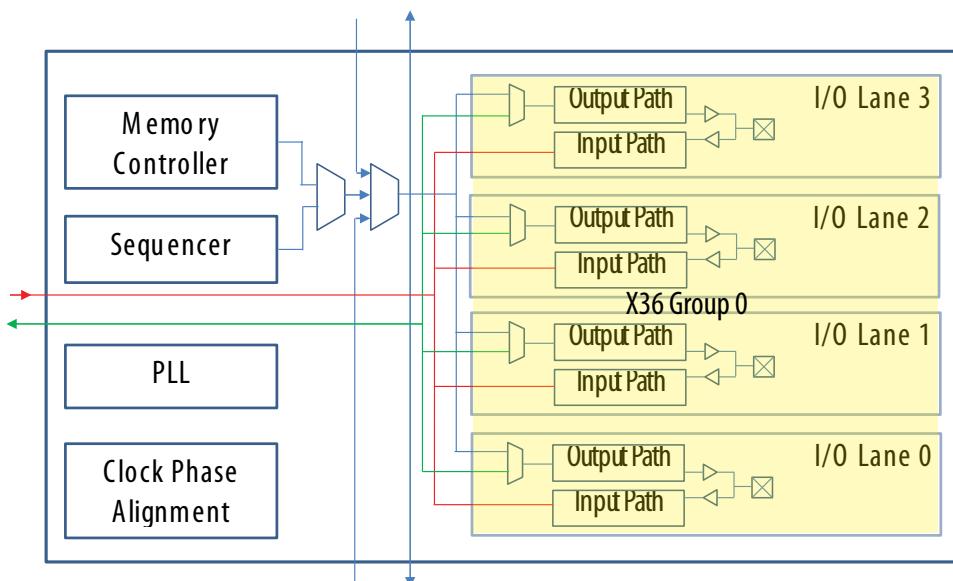
**Figure 86. x4 Group**

**Figure 87. x8 Group**




**Figure 88. x18 Group**



**Figure 89. x36 Group**



### 3.4.6. Arria 10 EMIF Architecture: Input DQS Clock Tree

The input DQS clock tree is a balanced clock network that distributes the read capture clock and strobe from the external memory device to the read capture registers inside the I/Os.

You can configure an input DQS clock tree in x4 mode, x8/x9 mode, x18 mode, or x36 mode.

Within every bank, only certain physical pins at specific locations can drive the input DQS clock trees. The pin locations that can drive the input DQS clock trees vary, depending on the size of the group.

**Table 42. Pins Usable as Read Capture Clock / Strobe Pair**

| Group Size | Index of Lanes Spanned by Clock Tree | In-Bank Index of Pins Usable as Read Capture Clock / Strobe Pair |              |
|------------|--------------------------------------|--|--------------|
|            |                                      | Positive Leg   | Negative Leg |
| x4         | 0A                                   | 4  | 5            |
| x4         | 0B                                   | 8  | 9            |
| x4         | 1A                                   | 16   | 17           |
| x4         | 1B                                   | 20   | 21           |
| x4         | 2A                                   | 28   | 29           |
| x4         | 2B                                   | 32   | 33           |
| x4         | 3A                                   | 40   | 41           |
| x4         | 3B                                   | 44   | 45           |
| x8 / x9    | 0                                    | 4  | 5            |
| x8 / x9    | 1                                    | 16   | 17           |
| x8 / x9    | 2                                    | 28   | 29           |
| x8 / x9    | 3                                    | 40   | 41           |
| x18        | 0, 1                                 | 12   | 13           |
| x18        | 2, 3                                 | 36   | 37           |
| x36        | 0, 1, 2, 3                           | 20   | 21           |

### 3.4.7. Arria 10 EMIF Architecture: PHY Clock Tree

Dedicated high-speed clock networks drive I/Os in Arria 10 EMIF. Each PHY clock network spans only one bank.

The relatively short span of the PHY clock trees results in low jitter and low duty-cycle distortion, maximizing the data valid window.

The PHY clock tree in Arria 10 devices can run as fast as 1.3 GHz. All Arria 10 external memory interfaces use the PHY clock trees.

### 3.4.8. Arria 10 EMIF Architecture: PLL Reference Clock Networks

Each I/O bank includes a PLL that can drive the PHY clock trees of that bank, through dedicated connections. In addition to supporting EMIF-specific functions, such PLLs can also serve as general-purpose PLLs for user logic.

Arria 10 external memory interfaces that span multiple banks use the PLL in each bank. (Previous device families relied on a single PLL with clock signals broadcast to all I/Os via a clock network.) The Arria 10 architecture allows for relatively short PHY clock networks, reducing jitter and duty-cycle distortion.

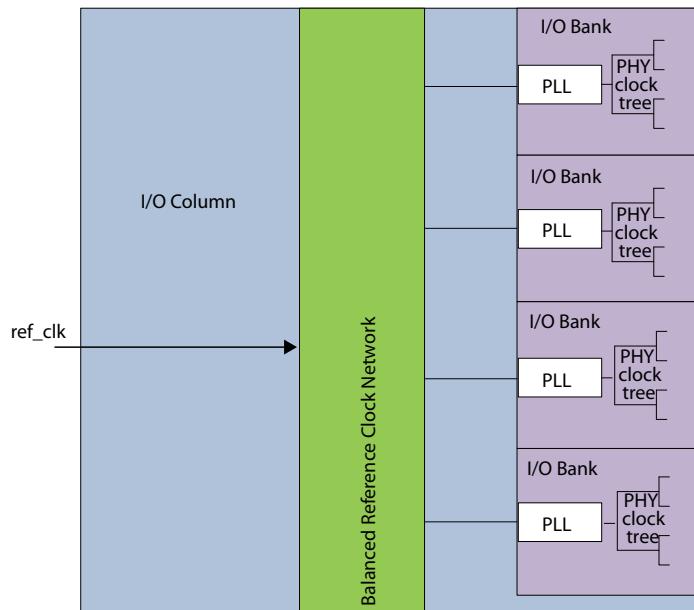


In a multi-bank interface, the clock outputs of individual PLLs must remain in phase; this is achieved by the following mechanisms:

- A single PLL reference clock source feeds all PLLs. The reference clock signal reaches the PLLs by a balanced PLL reference clock tree. The Quartus Prime software automatically configures the PLL reference clock tree so that it spans the correct number of banks.
- The IP sets the PLL M and N values appropriately to maintain synchronization among the clock dividers across the PLLs. This requirement restricts the legal PLL reference clock frequencies for a given memory interface frequency and clock rate. The Arria 10 EMIF IP parameter editor automatically calculates and displays the set of legal PLL reference clock frequencies. If you plan to use an on-board oscillator, you must ensure that its frequency matches the PLL reference clock frequency that you select from the displayed list. The correct M and N values of the PLLs are set automatically based on the PLL reference clock frequency that you select.

**Note:** The PLL reference clock pin may be placed in the address and command I/O bank or in a data I/O bank, there is no implication on timing.

**Figure 90. PLL Balanced Reference Clock Tree**

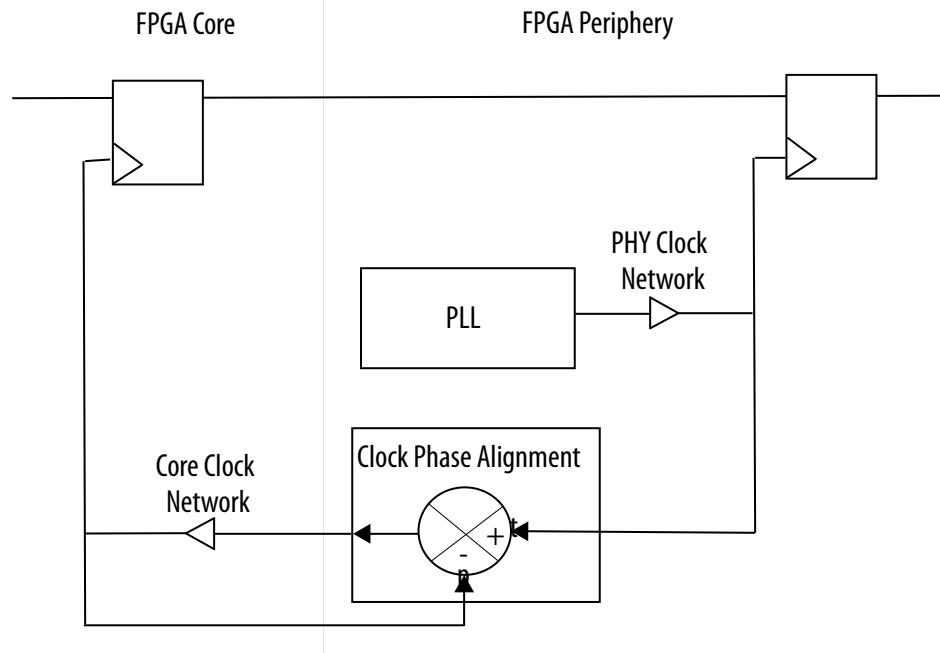


### 3.4.9. Arria 10 EMIF Architecture: Clock Phase Alignment

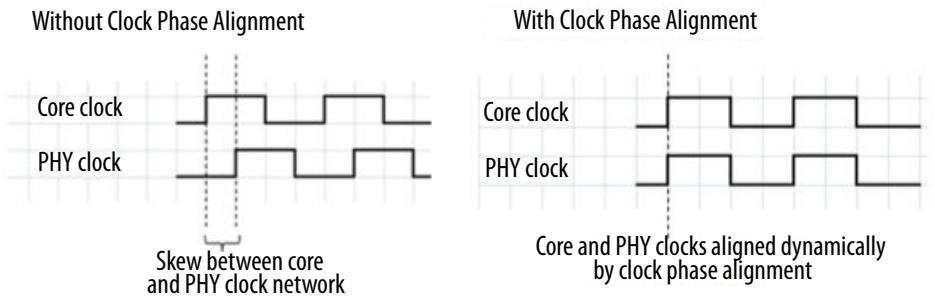
In Arria 10 external memory interfaces, a global clock network clocks registers inside the FPGA core, and the PHY clock network clocks registers inside the FPGA periphery. Clock phase alignment circuitry employs negative feedback to dynamically adjust the phase of the core clock signal to match the phase of the PHY clock signal.

The clock phase alignment feature effectively eliminates the clock skew effect in all transfers between the core and the periphery, facilitating timing closure. All Arria 10 external memory interfaces employ clock phase alignment circuitry.

**Figure 91. Clock Phase Alignment Illustration**



**Figure 92. Effect of Clock Phase Alignment**



## 3.5. Hardware Resource Sharing Among Multiple EMIFs

Often, it is necessary or desirable to share resources between interfaces.

The following topics explain which hardware resources can be shared, and provide guidance for doing so.

### 3.5.1. I/O Aux Sharing

The I/O Aux contains a hard Nios-II processor and dedicated memory storing the calibration software code and data.



When a column contains multiple memory interfaces, the hard Nios-II processor calibrates each interface serially. Interfaces placed within the same I/O column always share the same I/O Aux. The Quartus Prime Fitter handles I/O Aux sharing automatically.

### 3.5.2. I/O Bank Sharing

Data lanes from multiple compatible interfaces can share a physical I/O bank to achieve a more compact pin placement. To share an I/O bank, interfaces must use the same memory protocol, rate, frequency, I/O standard, and PLL reference clock signal.

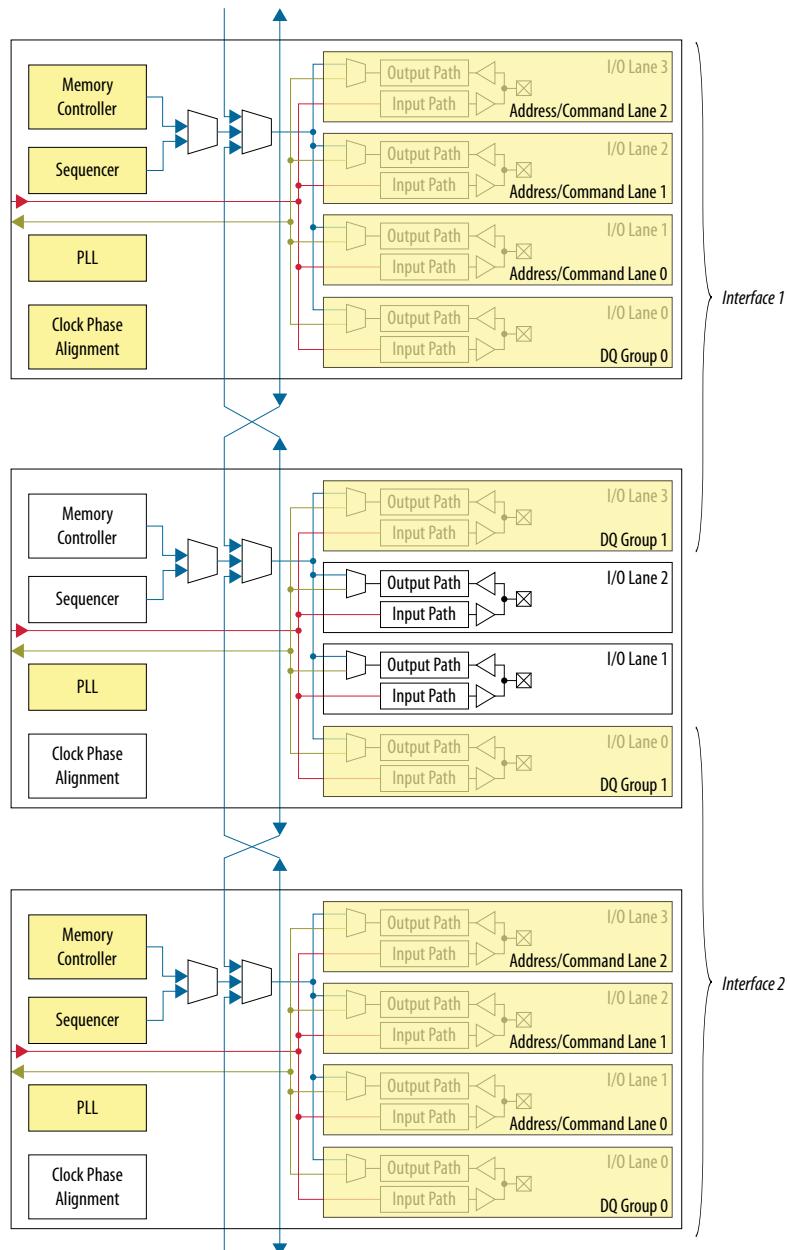
#### Rules for Sharing I/O Banks

- A bank cannot serve as the address and command bank for more than one interface. This means that lanes which implement address and command pins for different interfaces cannot be allocated to the same physical bank.

*Note:* An exception to the above rule exists when two interfaces are configured in a Ping-Pong PHY fashion. In such a configuration, two interfaces share the same set of address and command pins, effectively meaning that they share the same address and command tile.

- Pins within a lane cannot be shared by multiple memory interfaces.
- Pins that are not used by EMIF IP can serve as general-purpose I/Os of compatible voltage and termination settings.
- You can configure a bank as LVDS or as EMIF, but not both at the same time.
- Interfaces that share banks must reside at consecutive bank locations.

The following diagram illustrates two x16 interfaces sharing an I/O bank. The two interfaces share the same clock phase alignment block, so that one core clock signal can interact with both interfaces. Without sharing, the two interfaces would occupy a total of four physical banks instead of three.

**Figure 93. I/O Bank Sharing**


### 3.5.3. PLL Reference Clock Sharing

In Arria 10, every I/O bank contains a PLL, meaning that it is not necessary to share PLLs in the interest of conserving resources. Nonetheless, it is often desirable to share PLLs for other reasons.



You might want to share PLLs between interfaces for the following reasons:

- To conserve pins.
- When combined with the use of the balanced PLL reference clock tree, to allow the clock signals at different interfaces to be synchronous and aligned to each other. For this reason, interfaces that share core clock signals must also share the PLL reference clock signal.

To implement PLL reference clock sharing, open your RTL and connect the PLL reference clock signal at your design's top-level to the PLL reference clock port of multiple interfaces.

To share a PLL reference clock, the following requirements must be met:

- Interfaces must expect a reference clock signal of the same frequency.
- Interfaces must be placed in the same column.
- Interfaces must be placed at adjacent bank locations.

### 3.5.4. Core Clock Network Sharing

It is often desirable or necessary for multiple memory interfaces to be accessible using a single clock domain in the FPGA core.

You might want to share core clock networks for the following reasons:

- To minimize the area and latency penalty associated with clock domain crossing.
- To minimize consumption of core clock networks.

Multiple memory interfaces can share the same core clock signals under the following conditions:

- The memory interfaces have the same protocol, rate, frequency, and PLL reference clock source.
- The interfaces reside in the same I/O column.
- The interfaces reside in adjacent bank locations.

For multiple memory interfaces to share core clocks, you must specify one of the interfaces as master and the remaining interfaces as slaves. Use the `Core_clocks_sharing` setting in the parameter editor to specify the master and slaves.

In your RTL, connect the `clks_sharing_master_out` signal from the master interface to the `clks_sharing_slave_in` signal of all the slave interfaces. Both the master and slave interfaces expose their own output clock ports in the RTL (e.g. `emif_usr_clk`, `afi_clk`), but the signals are equivalent, so it does not matter whether a clock port from a master or a slave is used.

Core clock sharing necessitates PLL reference clock sharing; therefore, only the master interface exposes an input port for the PLL reference clock. All slave interfaces use the same PLL reference clock signal.



## 3.6. Arria 10 EMIF IP Component

The external memory interface IP component for Arria 10 provides a complete solution for implementing DDR3, DDR4, and QDR-IV external memory interfaces. The EMIF IP also includes a protocol-specific calibration algorithm that automatically determines the optimal delay settings for a robust external memory interface.

The external memory interface IP comprises the following parts:

- A set of synthesizable files that you can integrate into a larger design
- A stand-alone synthesizable example design that you can use for hardware validation
- A set of simulation files that you can incorporate into a larger project
- A stand-alone simulation example project that you can use to observe controller and PHY operation
- A set of timing scripts that you can use to determine the maximum operating frequency of the memory interface based on external factors such as board skew, trace delays, and memory component timing parameters
- A customized data sheet specific to your memory interface configuration

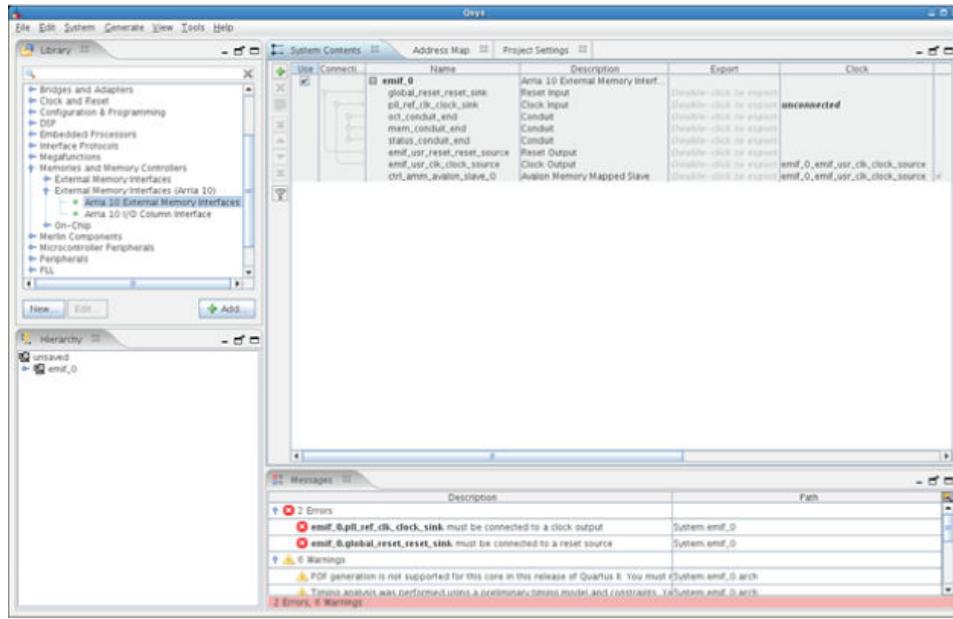
### 3.6.1. Instantiating Your Arria 10 EMIF IP in a Qsys Project

The following steps describe how to instantiate your Arria 10 EMIF IP in a Qsys project.

1. Within the Qsys interface, select **Memories and Memory Controllers** in the component **Library** tree.
2. Under **Memories and Memory Controllers**, select **External Memory Interfaces (Arria 10)**.
3. Under **External Memory Interfaces (Arria 10)**, select the **Arria 10 External Memory Interface** component.



**Figure 94. Instantiating Arria 10 EMIF IP in Qsys**



### 3.6.1.1. Logical Connections

The following logical connections exist in an Arria 10 EMIF IP core.

**Table 43. Logical Connections Table**

| Logical Connection                              | Description  |
|---|--|
| afi_conduit_end<br>(Conduit)                    | The Altera PHY Interface (AFI) connects a memory controller to the PHY. This interface is exposed only when you configure the memory interface in PHY-Only mode. The interface is synchronous to the afi_clk clock and afi_reset_n reset.  |
| afi_clk_conduit_end<br>(Conduit)                | Use this clock signal to clock the soft controller logic. The afi_clk is an output clock coming from the PHY when the memory interface is in PHY-Only mode. The phase of afi_clk is adjusted dynamically by hard circuitry for the best data transfer between FPGA core logic and periphery logic with maximum timing margin. Multiple memory interface instances can share a single afi_clk using the <b>Core Clocks Sharing</b> option during IP generation. |
| afi_half_clk_conduit_end<br>(Conduit)           | This clock runs at half the frequency of afi_clk. It is exposed only when the memory interface is in PHY-Only mode.  |
| afi_reset_n_conduit_end<br>(Conduit)            | This single-bit reset provides a synchronized reset output. Use this signal to reset all registers that are clocked by either afi_clk or afi_half_clk.   |
| cal_debug_avalon_slave<br>(Avalon Slave/Target) | This interface is exposed when the <b>EMIF Debug Toolkit/On-chip Debug Port</b> option is set to <b>Export</b> . This interface can be connected to an Arria 10 External Memory Interface Debug Component to allow EMIF Debug Toolkit access, or it can be used directly by user logic to access calibration diagnostic data.  |
| cal_debug_clk_clock_sink<br>(Clock Input)       | This clock is used for the cal_debug_avalon_slave interface. It can be connected to the emif_usr_clk_clock_source interface.   |
| cal_debug_reset_reset_sink<br>(Reset Input)     | This reset is used for the cal_debug_avalon_slave interface. It can be connected to the emif_usr_reset_reset_source interface.   |

*continued...*



| Logical Connection                                    | Description   |
|---|---|
| cal_debug_out_avalon_master<br>(Avalon Master/Source) | This interface is exposed when the <b>Enable Daisy-Chaining for EMIF Debug Toolkit/On-Chip Debug Port</b> option is enabled. Connect this interface to the cal_debug_avalon_slave interface of the next EMIF instance in the same I/O column.   |
| cal_debug_out_clk_clock_source<br>(Clock Output)      | This interface should be connected to the cal_debug_clk_clock_sink interface of the next EMIF instance in the same I/O column (similar to cal_debug_out_avalon_master).   |
| cal_debug_out_reset_reset_source<br>(Reset Output)    | This interface should be connected to the cal_debug_reset_reset_sink interface of the next EMIF instance in the same I/O column (similar to cal_debug_out_avalon_master).   |
| effmon_csr_avalon_slave<br>(Avalon Slave/Target)      | This interface allows access to the Efficiency Monitor CSR. For more information, see the documentation on the UniPHY Efficiency Monitor.   |
| global_reset_reset_sink<br>(Reset Input)              | This single-wire input port is the asynchronous reset input for the EMIF core.  |
| pll_ref_clk_clock_sink<br>(Clock Input)               | This single-wire input port connects the external PLL reference clock to the EMIF core. Multiple EMIF cores may share a PLL reference clock source, provided the restrictions outlined in the PLL and PLL Reference Clock Network section are observed.   |
| oct_conduit_end<br>(Conduit)                          | This logical port is connected to an OCT pin and provides calibrated reference data for EMIF cores with pins that use signaling standards that require on-chip termination. Depending on the I/O standard, reference voltage, and memory protocol, multiple EMIF cores may share a single OCT pin.  |
| mem_conduit_end<br>(Conduit)                          | This logical conduit can attach an Altera Memory Model to an EMIF core for simulation. Memory models for various protocols are available under the <b>Memories and Memory Controllers — External Memory Interfaces — Memory Models</b> section of the component library in Qsys. You must ensure that all configuration parameters for the memory model match the configuration parameters of the EMIF core.  |
| status_conduit_end<br>(Conduit)                       | The status conduit exports two signals that can be sampled to determine if the calibration operation passed or failed for that core.  |
| emif_usr_reset_reset_source<br>(Reset Output)         | This single-bit reset output provides a synchronized reset output that should be used to reset all components that are synchronously connected to the EMIF core. Assertion of the global reset input triggers an assertion of this output as well, therefore you should rely on this signal only as a reset source for all components connected to the EMIF core.   |
| emif_usr_clk_clock_source<br>(Clock Output)           | Use this single-bit clock output to clock all logic connected to the EMIF core. The phase of this clock signal is adjusted dynamically by circuitry in the EMIF core such that data can be transferred between core logic and periphery registers with maximum timing margin. Drive all logic connected to the EMIF core with this clock signal. Other clock sources generated from the same reference clock or even the same PLL may have unknown phase relationships. Multiple EMIF cores can share a single core clock using the Core Clocks Sharing option described in the Example Design tab of the parameter editor.   |
| ctrl_amm_avalon_slave<br>(Avalon Slave/Target)        | This Avalon target port initiates read or write commands to the controller. Refer to the Avalon Interface Specification for more information on how to design cores that comply to the Avalon Bus Specification.<br>For DDR3, DDR4, and LPDDR3 protocols with the hard PHY and hard controller configuration and an AVL slave interface exposed, ctrl_amm_avalon_slave is renamed to ctrl_amm_avalon_slave_0.<br>For QDR II, QDR II+, and QDR II+ Xtreme interfaces with hard PHY and soft controller, separate read and write connections are used.<br>ctrl_amm_avalon_slave_0 is the read port and ctrl_amm_avalon_slave_1 is the write port.<br>For QDR-IV interfaces with hard PHY and soft controller operating at quarter rate, a total of eight separate Avalon interfaces (named ctrl_amm_avalon_slave_0 to ctrl_amm_avalon_slave_7) are used to maximize bus efficiency. |



### 3.6.2. File Sets

The Arria 10 EMIF IP core generates four output file sets for every EMIF IP core, arranged according to the following directory structure.

**Table 44. Generated File Sets**

| Directory                            | Description   |
|--------------------------------------|---|
| <core_name>/*                        | This directory contains only the files required to integrate a generated EMIF core into a larger design. This directory contains: <ul style="list-style-type: none"><li>• Synthesizable HDL source files</li><li>• Customized TCL timing scripts specific to the core (protocol and topology)</li><li>• HEX files used by the calibration algorithm to identify the interface parameters</li><li>• A customized data sheet that describes the operation of the generated core</li></ul> <p><i>Note:</i> The top-level HDL file is generated in the root folder as &lt;core_name&gt;.v (or &lt;core_name&gt;.vhd for VHDL designs). You can reopen this file in the parameter editor if you want to modify the EMIF core parameters and regenerate the design.</p> |
| <core_name>_sim/*                    | This directory contains the simulation fileset for the generated EMIF core. These files can be integrated into a larger simulation project. For convenience, simulation scripts for compiling the core are provided in the /mentor, /cadence, /synopsys, and /riviera subdirectories.<br><br>The top-level HDL file, <core_name>.v (or <core_name>.vhd) is located in this folder, and all remaining HDL files are placed in the /altera_emif_arch_nf subfolder, with the customized data sheet. The contents of this directory are not intended for synthesis.   |
| emif_<instance_num>_example_design/* | This directory contains a set of TCL scripts, QSYS project files and README files for the complete synthesis and simulation example designs. You can invoke these scripts to generate a standalone fully-synthesizable project complete with an example driver, or a standalone simulation design complete with an example driver and a memory model.   |

### 3.6.3. Customized readme.txt File

When you generate your Arria 10 EMIF IP, the system produces a customized readme file, containing data indicative of the settings in your IP core.

The readme file is <variation\_name>/altera\_emif\_arch\_nf\_<version\_number>/<synth/sim>/<variation\_name>\_altera\_emif\_arch\_nf\_<version\_number>\_<unique ID>\_readme.txt, and contains a summary of Arria 10 EMIF information, and details specific to your IP core, including:

- Pin location guidelines
- External port names, directions, and widths
- Internal port names, directions, and widths
- Avalon interface configuration details (if applicable)
- Calibration mode
- A brief summary of all configuration settings for the generated IP



You should review the generated readme file for implementation guidelines specific to your IP core.

### 3.6.4. Clock Domains

The Arria 10 EMIF IP core provides a single clock domain to drive all logic connected to the EMIF core.

The frequency of the clock depends on the core-clock to memory-clock interface rate ratio. For example, a quarter-rate interface with an 800 MHz memory clock would provide a 200 MHz clock to the core ( $800 \text{ MHz} / 4 = 200 \text{ MHz}$ ). The EMIF IP dynamically adjusts the phase of the core clock with respect to the periphery clock to maintain the optimum alignment for transferring data between the core and periphery. Independent EMIF IP cores driven from the same reference clock have independent core clock domains. You should employ one of the following strategies if you are implementing multiple EMIF cores:

1. Treat all crossing between independent EMIF-generated clock domains as asynchronous, even though they are generated from the same reference clock.
2. Use the **Core clock sharing** option to enforce that multiple EMIF cores share the same core clock. You must enable this option during IP generation. This option is applicable only for cores that reside in the same I/O column.

### 3.6.5. ECC in Arria 10 EMIF IP

The ECC (error correction code) is a soft component of the Arria 10 EMIF IP that reduces the chance of errors when reading and writing to external memory. ECC allows correction of single-bit errors and reduces the chances of system failure.

The ECC component includes an encoder, decoder, write FIFO buffer, and modification logic, to allow read-modify-write operations. The ECC code employs standard Hamming logic to correct single-bit errors and to detect double-bit errors. ECC is available in 16, 24, 40, and 72 bit widths.

When writing data to memory, the encoder creates ECC bits and writes them together with the regular data. When reading from memory, the decoder checks the ECC bits and regular data, and passes the regular data unchanged if no errors are detected. If a single-bit error is detected, the ECC logic corrects the error and passes the regular data. If more than a single-bit error is detected, the ECC logic sets a flag to indicate the error.

Read-modify-write operations can occur in the following circumstances:

- A partial write in data mask mode with ECC enabled, where at least one memory burst of byte-enable is not all ones or all zeros.
- Auto-correction with ECC enabled. This is usually a dummy write issued by the auto-correction logic to correct the memory content when a single-bit error is detected. The read-modify-write reads back the data, corrects the single-bit error, and writes the data back.

The additional overhead associated with read-modify-write operations can severely reduce memory interface efficiency. For best efficiency, you should design traffic patterns to avoid read-modify-write operations wherever possible, such as by minimizing the number of partial writes in ECC mode.



### 3.6.5.1. ECC Components

The ECC logic communicates with user logic via an Avalon-MM interface, and with the hard memory controller via an Avalon-ST interface

#### ECC Encoder

The ECC encoder consists of a x64/x72 encoder IP core capable of single-bit error correction and double-bit error detection. The encoder takes 64 bits input and converts it to 72 bits output, where the 8 additional bits are ECC code. The encoder supports any input data width less than 64-bits. Any unused input data bits are set to zero.

#### ECC Decoder

The ECC decoder consists of a x72/x64 decoder IP core capable of double-bit error detection. The decoder takes 72 bits input and converts it to 64 bits output. The decoder also produces single-bit error and double-bit error information. The decoder controls the user read data valid signal; when read data is intended for partial write, the user read data valid signal is deasserted, because the read data is meant for merging, not for the user.

#### Partial Write Data FIFO Buffer

The Partial Write Data FIFO Buffer is implemented in soft logic to store partial write data and byte enable. Data and byte enable are popped and merged with the returned read data. A partial write can occur in the following situations:

- At least one memory burst of byte enable is not all ones or all zeroes.
- Non data masked mode, where all memory bursts of byte enable are not all ones.
- A dummy write with auto-correction logic, where all memory bursts of byte enable are all zeroes. (You might use a dummy write when correcting memory content with a single-bit error.)

#### Merging Logic

Merge return partial read data with write data based on byte enabled popped from the FIFO buffer, and send it to the ECC encoder.

#### Pointer FIFO Buffer

The pointer FIFO buffer is implemented in soft logic to store write data pointers. The ECC logic refers to the pointers when sending write data to DBC. The pointers serve to overwrite existing write data in the data buffer during a read-modify-write process.

#### Partial Logic

Partial logic decodes byte enable information and distinguishes between normal and partial writes.

#### Memory Mode Register Interface

The Memory Mode Register interface is an Avalon-based interface through which core logic can access debug signals and sideband operation requests in the hard memory controller.

The MMR logic routes ECC-related operations to an MMR register implemented in soft logic, and returns the ECC information via an Avalon-MM interface. The MMR logic tracks single-bit and double-bit error status, and provides the following information:

- Interrupt status.
- Single-bit error and double-bit error status.
- Single-bit error and double-bit error counts (to a maximum of 15; if more than 15 errors occur, the count will overflow).
- Address of the last error.

### 3.6.5.2. ECC User Interface Controls

You can enable the ECC logic from the **Configuration, Status, and Error Handling** section of the **Controller** tab in the parameter editor.

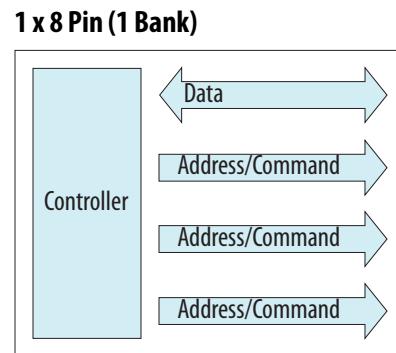
There are three user interface settings related to ECC:

- **Enabled Memory-Mapped Configuration and Status Register (MMR) Interface:** Allows run-time configuration of the memory controller. You can enable this option together with ECC to retrieve error detection information from the ECC logic.
- **Enabled Error Detection and Correction Logic:** Enables ECC logic for single-bit error correction and double-bit error detection.
- **Enable Auto Error Correction:** Allows the controller to automatically correct single-bit errors detected by the ECC logic.

## 3.7. Examples of External Memory Interface Implementations for DDR4

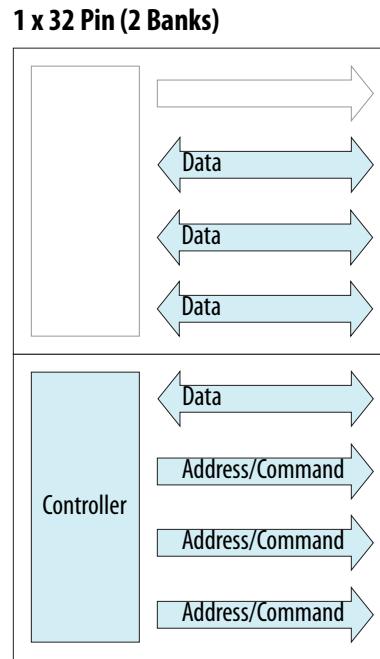
The following figures are examples of external memory interface implementations for different DDR4 memory widths. The figures show the locations of the address/command and data pins in relation to the locations of the memory controllers.

**Figure 95. DDR4 1x8 Implementation Example (One I/O Bank)**

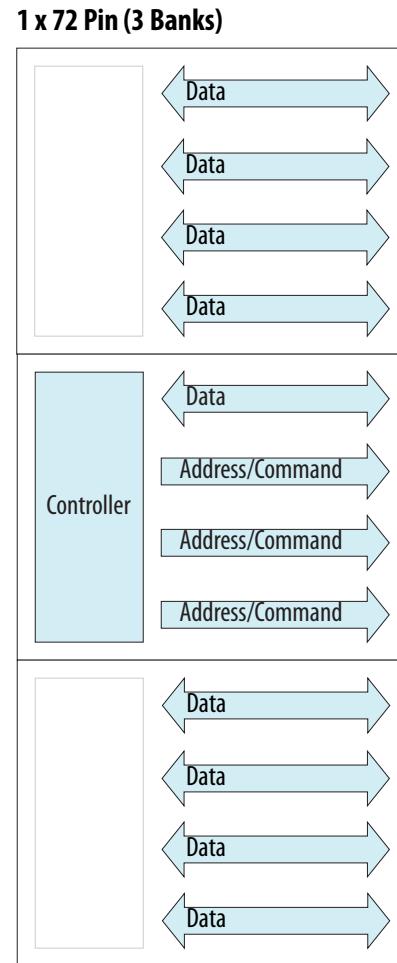




**Figure 96. DDR4 1x32 Implementation Example (Two I/O Banks)**



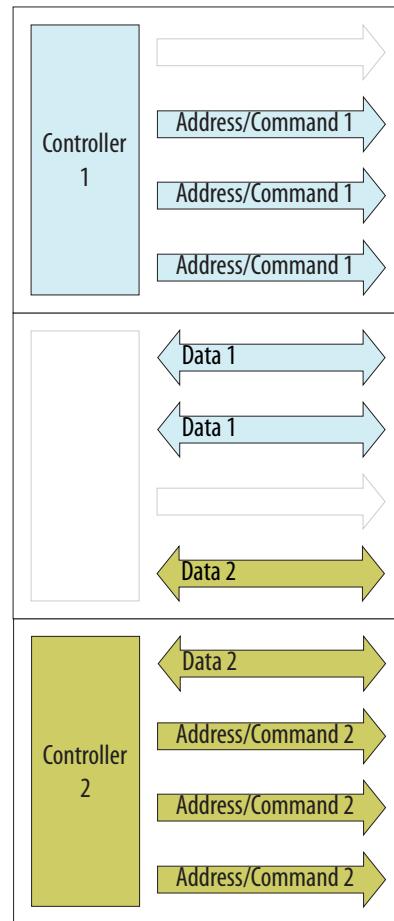
**Figure 97. DDR4 1x72 Implementation Example (Three I/O Banks)**



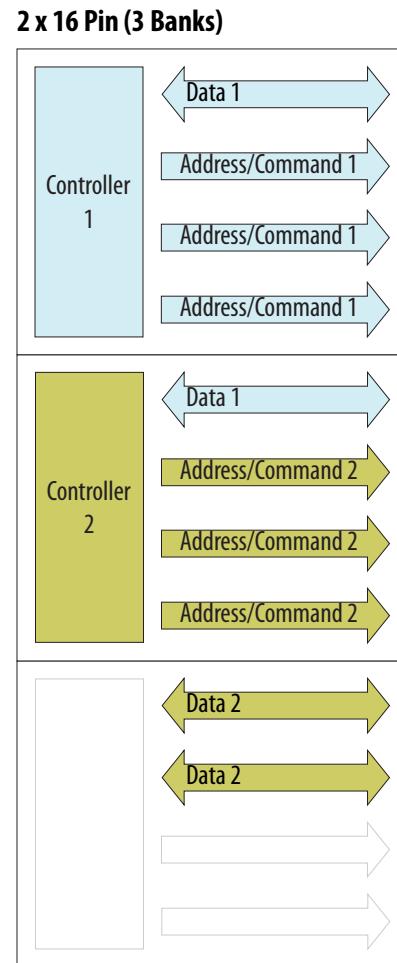


**Figure 98. DDR4 2x16 Implementation Example with Controllers in Non-Adjacent Banks (Three I/O Banks)**

**2 x 16 Pin (3 Banks)**

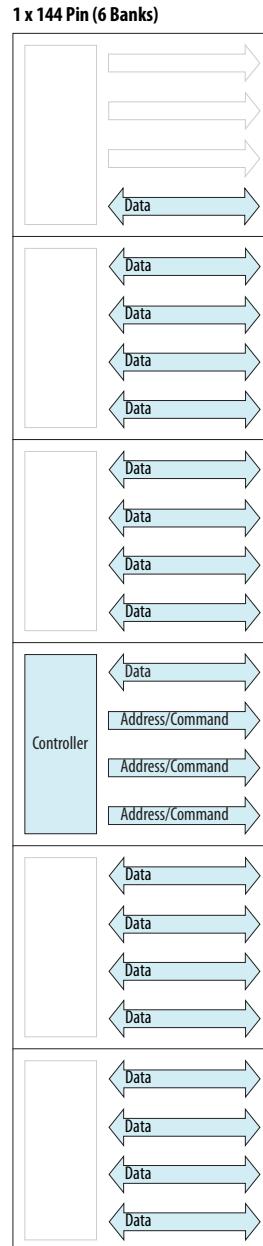


**Figure 99. DDR4 2x16 Implementation Example with Controllers in Adjacent Banks (Three I/O Banks)**





**Figure 100. DDR4 1x144 Implementation Example (Six I/O Banks)**



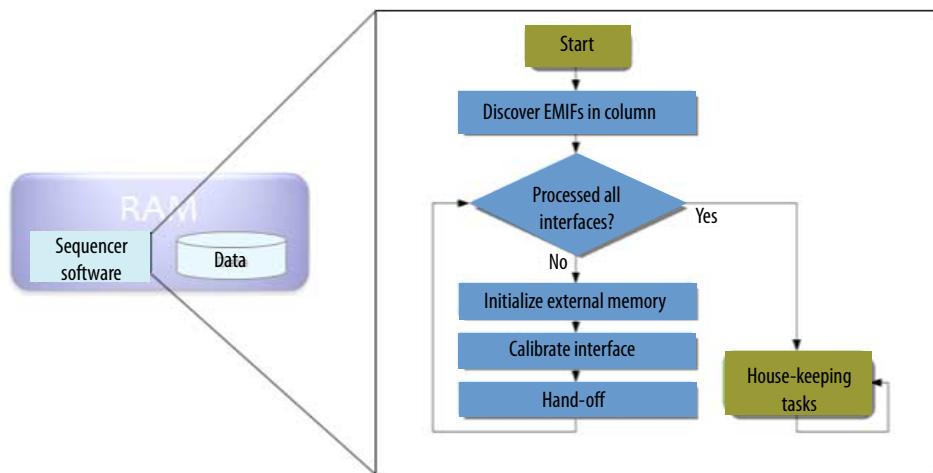
### 3.8. Arria 10 EMIF Sequencer

The Arria 10 EMIF sequencer is fully hardened in silicon, with executable code to handle protocols and topologies. Hardened RAM contains the calibration algorithm.

The Arria 10 EMIF sequencer is responsible for the following operations:

- Initializes memory devices.
- Calibrates the external memory interface.
- Governs the hand-off of control to the memory controller.
- Handles recalibration requests and debug requests.
- Handles all supported protocols and configurations.

**Figure 101. Arria 10 EMIF Sequencer Operation**



### 3.8.1. Arria 10 EMIF DQS Tracking

DQS tracking is enabled for QDR II / II+ / QDR II+ Xtreme, RLDRAM 3, and LPDDR3 protocols. DQS tracking is not available for DDR3 and DDR4 protocols.

## 3.9. Arria 10 EMIF Calibration

The calibration process compensates for skews and delays in the external memory interface.

The calibration process enables the system to compensate for the effects of factors such as the following:

- Timing and electrical constraints, such as setup/hold time and Vref variations.
- Circuit board and package factors, such as skew, fly-by effects, and manufacturing variations.
- Environmental uncertainties, such as variations in voltage and temperature.
- The demanding effects of small margins associated with high-speed operation.

### 3.9.1. Calibration Stages

At a high level, the calibration routine consists of address and command calibration, read calibration, and write calibration.



The stages of calibration vary, depending on the protocol of the external memory interface.

**Table 45. Calibration Stages by Protocol**

| Stage                      | DDR4 | DDR3 | LPDDR3 | RLDRAM II/3 | QDR-IV | QDR II/II+ |
|----------------------------|------|------|--------|-------------|--------|------------|
| <b>Address and command</b> |      |      |        |             |        |            |
| Leveling                   | Yes  | Yes  | —      | —           | —      | —          |
| Deskew                     | Yes  | —    | Yes    | —           | Yes    | —          |
| <b>Read</b>                |      |      |        |             |        |            |
| DQSen                      | Yes  | Yes  | Yes    | Yes         | Yes    | Yes        |
| Deskew                     | Yes  | Yes  | Yes    | Yes         | Yes    | Yes        |
| VREF-In                    | Yes  | —    | —      | —           | Yes    | —          |
| LFIFO                      | Yes  | Yes  | Yes    | Yes         | Yes    | Yes        |
| <b>Write</b>               |      |      |        |             |        |            |
| Leveling                   | Yes  | Yes  | Yes    | Yes         | Yes    | —          |
| Deskew                     | Yes  | Yes  | Yes    | Yes         | Yes    | Yes        |
| VREF-Out                   | Yes  | —    | —      | —           | —      | —          |

### 3.9.2. Calibration Stages Descriptions

The various stages of calibration perform address and command calibration, read calibration, and write calibration.

#### Address and Command Calibration

The goal of address and command calibration is to delay address and command signals as necessary to optimize the address and command window. This stage is not available for all protocols, and cannot compensate for an inefficient board design.

Address and command calibration consists of the following parts:

- Leveling calibration— Centers the CS# signal and the entire address and command bus, relative to the CK clock. This operation is available only for DDR3 and DDR4 interfaces.
- Deskew calibration— Provides per-bit deskew for the address and command bus (except CS#), relative to the CK clock. This operation is available for DDR4 and QDR-IV interfaces only.



### Read Calibration

Read calibration consists of the following parts:

- DQSen calibration— Calibrates the timing of the read capture clock gating and ungating, so that the PHY can gate and ungate the read clock at precisely the correct time—if too early or too late, data corruption can occur. The algorithm for this stage varies, depending on the memory protocol.
- Deskew calibration— Performs per-bit deskew of read data relative to the read strobe or clock.
- VREF-IN calibration— Calibrates the Vref level at the FPGA.
- LFIFO calibration: Normalizes differences in read delays between groups due to fly-by, skews, and other variables and uncertainties.

### Write Calibration

Write calibration consists of the following parts:

- Leveling calibration— Aligns the write strobe and clock to the memory clock, to compensate for skews, especially those associated with fly-by topology. The algorithm for this stage varies, depending on the memory protocol.
- Deskew calibration— Performs per-bit deskew of write data relative to the write strobe and clock.
- VREF-Out calibration— Calibrates the VREF level at the memory device.

### 3.9.3. Calibration Algorithms

The calibration algorithms sometimes vary, depending on the targeted memory protocol.

#### Address and Command Calibration

Address and command calibration consists of the following parts:



- Leveling calibration— (DDR3 and DDR4 only) Toggles the CS# and CAS# signals to send read commands while keeping other address and command signals constant. The algorithm monitors for incoming DQS signals, and if the DQS signal toggles, it indicates that the read commands have been accepted. The algorithm then repeats using different delay values, to find the optimal window.
- Deskew calibration— (DDR4, QDR-IV, and LPDDR3 only)
  - (DDR4) Uses the DDR4 address and command parity feature. The FPGA sends the address and command parity bit, and the DDR4 memory device responds with an alert signal if the parity bit is detected. The alert signal from the memory device tells the FPGA that the parity bit was received.  
Deskew calibration requires use of the PAR/ALERT# pins, so you should not omit these pins from your design. One limitation of deskew calibration is that it cannot deskew ODT and CKE pins.
  - (QDR-IV) Uses the QDR-IV loopback mode. The FPGA sends address and command signals, and the memory device sends back the address and command signals which it captures, via the read data pins. The returned signals indicate to the FPGA what the memory device has captured. Deskew calibration can deskew all synchronous address and command signals.
  - (LPDDR3) Uses the LPDDR3 CA training mode. The FPGA sends signals onto the LPDDR3 CA bus, and the memory device sends back those signals that it captures, via the DQ pins. The returned signals indicate to the FPGA what the memory device has captured. Deskew calibration can deskew all signals on the CA bus. The remaining command signals (CS, CKE, and ODT) are calibrated based on the average of the deskewed CA bus.



## Read Calibration

- DQSen calibration— (DDR3, DDR4, LPDDR3, RLDRAMx and QDRx) DQSen calibration occurs before Read deskew, therefore only a single DQ bit is required to pass in order to achieve a successful read pass.
  - (DDR3, DDR4, and LPDDR3) The DQSen calibration algorithm searches the DQS preamble using a hardware state machine. The algorithm sends many back-to-back reads with a one clock cycle gap between. The hardware state machine searches for the DQS gap while sweeping DQSen delay values. The algorithm then increments the VFIFO value, and repeats the process until a pattern is found. The process is then repeated for all other read DQS groups.
  - (RLDRAMx and QDRx) The DQSen calibration algorithm does not use a hardware state machine; rather, it calibrates cycle-level delays using software and subcycle delays using DQS tracking hardware. The algorithm requires good data in memory, and therefore relies on guaranteed writes. (Writing a burst of 0s to one location, and a burst of 1s to another; back-to-back reads from these two locations are used for read calibration.)

The algorithm enables DQS tracking to calibrate the phase component of DQS enable. It then issues a guaranteed write, followed by back-to-back reads. The algorithm sweeps DQSen values cycle by cycle until the read operation succeeds. The process is then repeated for all other read groups.

- Deskew calibration— Read deskew calibration is performed before write leveling, and must be performed at least twice: once before write calibration, using simple data patterns from guaranteed writes, and again after write calibration, using complex data patterns.
- The deskew calibration algorithm performs a guaranteed write, and then sweeps `dqs_in` delay values from low to high, to find the right edge of the read window. The algorithm then sweeps `dq_in` delay values low to high, to find the left edge of the read window. Updated `dqs_in` and `dq_in` delay values are then applied to center the read window. The algorithm then repeats the process for all data pins.
- Vref-In calibration— Read Vref-In calibration begins by programming Vref-In with an arbitrary value. The algorithm then sweeps the Vref-In value from the starting value to both ends, and measures the read window for each value. The algorithm selects the Vref-In value which provides the maximum read window.
  - LFIFO calibration— Read LFIFO calibration normalizes read delays between groups. The PHY must present all data to the controller as a single data bus. The LFIFO latency should be large enough for the slowest read data group, and large enough to allow proper synchronization across FIFOs.



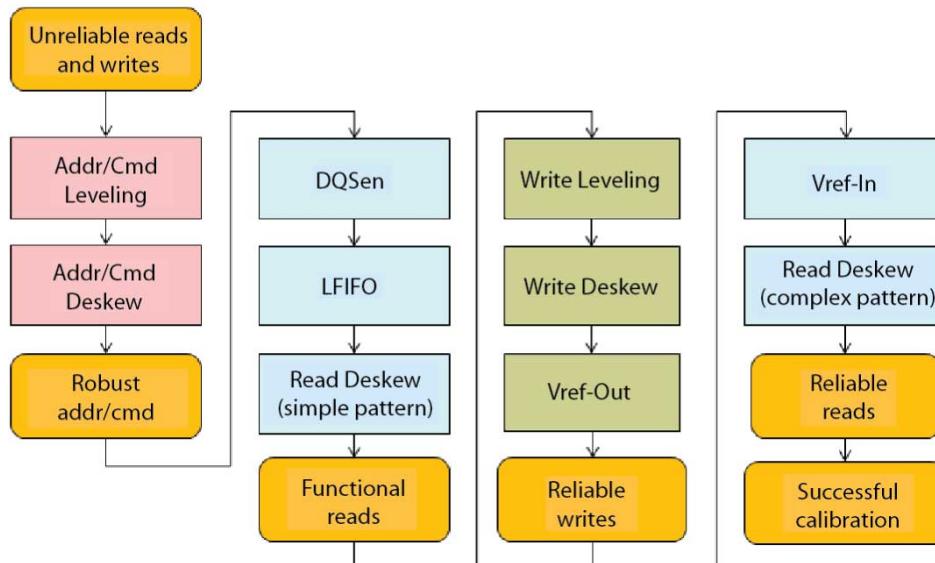
## Write Calibration

- Leveling calibration— Write leveling calibration aligns the write strobe and clock to the memory clock, to compensate for skews. In general, leveling calibration tries a variety of delay values to determine the edges of the write window, and then selects an appropriate value to center the window. The details of the algorithm vary, depending on the memory protocol.
  - (DDR $x$ , LPDDR3) Write leveling occurs before write deskew, therefore only one successful DQ bit is required to register a pass. Write leveling staggers the DQ bus to ensure that at least one DQ bit falls within the valid write window.
  - (RLDRAM $x$ ) Optimizes for the CK versus DK relationship.
  - (QDR-IV) Optimizes for the CK versus DK relationship. Is covered by address and command deskew using the loopback mode.
  - (QDR II/II+/Xtreme) The K clock is the only clock, therefore write leveling is not required.
- Deskew calibration— Performs per-bit deskew of write data relative to the write strobe and clock. Write deskew calibration does not change dqs\_out delays; the write clock is aligned to the CK clock during write leveling.
- VREF-Out calibration— (DDR4) Calibrates the VREF level at the memory device. The VREF-Out calibration algorithm is similar to the VREF-In calibration algorithm.

### 3.9.4. Calibration Flowchart

The following flowchart illustrates the calibration flow.

**Figure 102. Calibration Flowchart**





### 3.9.5. Periodic OCT Recalibration

Periodic OCT recalibration improves the accuracy of the on-chip termination values used by DDR4 Pseudo-open Drain (POD) I/Os. This feature periodically invokes the user-mode OCT calibration engine and updates the I/O buffer termination settings to compensate for variations in calibrated OCT settings caused by large changes in device operating temperature.

This feature is automatically enabled for DDR4 memory interfaces unless the IP does not meet the technical requirements, or if you explicitly disable the feature in the parameter editor.

#### 3.9.5.1. Operation

The Periodic OCT recalibration engine refreshes the calibrated OCT settings for DDR4 I/O buffers every 500ms. To ensure data integrity, there is a momentary pause in user traffic as the OCT settings are refreshed; however, the process of OCT calibration is decoupled from the actual update to the I/O buffers, to minimize disruption of user traffic.

The calibration process uses the external RZQ reference resistor to determine the optimal settings for the I/O buffer, to meet the specified calibrated I/O standards on the FPGA. OCT Calibration only affects the I/O pin that is connected to the RZQ resistor; therefore, memory traffic is not interrupted during the calibration phase.

Upon completion of the calibration process, the updated calibration settings are applied to the I/O buffers. The memory traffic is halted momentarily by placing the memory into self-refresh mode; this ensures that the data bus is idle and no glitches are created by the I/O buffers during the buffer update. The buffer is updated as soon as the memory enters self-refresh mode. The memory interface exits self-refresh mode when the buffer update is complete and new read or write requests are detected on the Avalon bus. The controller remains in self-refresh mode until a new command is detected. OCT calibration continues to occur even if the memory is still in self refresh mode. Upon detection of a new command, the controller issues a self-refresh exit command to the memory, followed by a memory-side ZQ calibration short duration (ZQCS) command. Memory traffic resumes when the memory DLL has re-locked.

If you disable the periodic OCT recalibration engine, the calibration process occurs only once during device configuration. In this operating mode, the calibrated OCT settings can vary across temperature as specified by the calibration accuracy ranges listed in the Arria 10 Device Handbook. The DDR external timing report automatically factors in the effect of enabling or disabling the periodic OCT recalibration engine when calculating the total amount of external I/O transfer margin.

#### 3.9.5.2. Technical Restrictions

Certain criteria must be met in order to use periodic OCT recalibration.



The periodic OCT recalibration engine is enabled only when all of the following criteria are met:

- The memory interface is configured to use the Altera Hard Memory Controller for DDR4.
- The memory interface is configured for either DDR4 UDIMM or component topologies. RDIMM and LRDIMM topologies are not supported.
- The memory interface is not used with the hardened processor subsystem.
- The memory interface does not use Ping-Pong PHY.
- The memory interface does not use calibrated I/O standards for address, command, or clock signals.
- The memory interface uses calibrated I/O standards for the data bus.
- The memory does not use the memory mapped register (MMR) interface of the HMC, including ECC modes.
- You have not explicitly disabled periodic OCT recalibration in the parameter editor.
- The specified device is a production level device (that is, not an ES/ES2/ES3 class device).

Periodic OCT recalibration requires that each EMIF instance in the design employ a dedicated RZQ resistor. Because this restriction cannot be detected at IP generation time, you must explicitly disable the periodic OCT recalibration engine for a given interface if it shares an RZQ resistor with another interface. Ensure that you observe this restriction when automatically upgrading EMIF IP from older versions of the Quartus Prime software.

### 3.9.5.3. Efficiency Impact

The Periodic OCT recalibration engine must interrupt user traffic for a short period of time in order to update I/O buffer termination settings.

The exact flow of operations executed by the recalibration engine that affects memory traffic is described below:

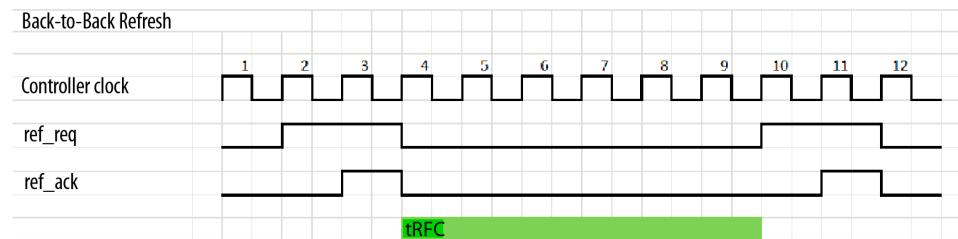
1. Enter Self-Refresh Mode. The EMIF calibration CPU triggers self-refresh entry on the hard memory controller. The controller flushes all pending operations, precharges all banks and issues the self-refresh command. This operation introduces a delay of approximately 25 Memory clock cycles (precharge all and self-refresh entry commands).
2. Confirm Self-Refresh Mode. The EMIF calibration CPU polls the hard memory controller to confirm that the clocks have stopped. This operation introduces no delay.
3. Issue codeword update. The EMIF calibration CPU triggers user-mode OCT logic to update code words. This operation introduces a delay of 50-100ns, depending on the device speed grade.

4. Allow Exit Self-Refresh Mode. The EMIF calibration CPU enables automatic self-refresh exit logic. This operation introduces a delay of 50-100ns, depending on the device speed grade.
5. Wait for Memory Traffic. The hard memory controller waits for an incoming read or write command on the Avalon bus. The delay introduced by this operation varies, depending on the user application.
6. Exit Self Refresh Mode. The hard memory controller issues the Self-Refresh Exit command and a simultaneous memory-side RZQ calibration (ZQCS) command. The delay introduced by this operation varies according to the device speed bin (up to ~1000 memory clock cycles for fastest memory devices).

The efficiency impact on throughput-sensitive work loads is less than one percent, even under worst-case scenarios with all banks active. However, be aware that the first command issued after the hard memory controller exits self-refresh mode will incur the latency overhead of waiting for the memory DLL to re-lock when the Self-Refresh Exit command is issued by the hard memory controller. Contact Intel FPGA Technical Services for information on how to manually trigger or inhibit periodic OCT updates for applications that are sensitive to latency.

### **3.10. Back-to-Back User-Controlled Refresh Usage in Arria 10**

The following diagram illustrates the back-to-back refresh model for optimized hard memory controller (HMC) performance in Arria 10 devices.



For optimal performance, ensure that you deassert the Refresh request after receiving the acknowledgement pulse. You can implement a timer to track  $t_{RFC}$  before asserting the next Refresh request. Failure to deassert the Refresh request can delay memory access to the rank not in refresh.

### **3.11. Arria 10 EMIF and SmartVID**

Arria 10 EMIF IP can be used with the SmartVID voltage management system, to achieve reduced power consumption.

*Note:* Arria 10 HPS EMIF IP does not currently support SmartVID.

The SmartVID controller allows the FPGA to operate at a reduced  $V_{CC}$ , while maintaining performance. Because the SmartVID controller can adjust  $V_{CC}$  up or down in response to power requirements and temperature, it can have an impact on external memory interface performance. When used with the SmartVID controller, the EMIF IP implements a handshake protocol to ensure that EMIF calibration does not begin until after voltage adjustment has completed.



In extended speed grade devices, voltage adjustment occurs once when the FPGA is powered up, and no further voltage adjustments occur. The external memory calibration occurs after this initial voltage adjustment is completed. EMIF specifications are expected to be slightly lower in extended speed grade devices using SmartVID, than in devices not using SmartVID.

In industrial speed grade devices, voltage adjustment occurs at power up, and may also occur during operation, in response to temperature changes. External memory interface calibration does not occur until after the initial voltage adjustment at power up. However, the external memory interface is not recalibrated in response to subsequent voltage adjustments that occur during operation. As a result, EMIF specifications for industrial speed grade devices using SmartVID are expected to be lower than for extended speed grade devices.

### Using Arria 10 EMIF IP with SmartVID

To employ Arria 10 EMIF IP with SmartVID, follow these steps:

1. Ensure that the Quartus Prime project and Qsys system are configured to use VID components. This step exposes the `vid_cal_done_persist` interface on instantiated EMIF IP, which is required for communicating with the SmartVID controller.
2. Instantiate the SmartVID controller, using an I/O PLL IP core to drive the 125MHz `vid_clk` and the 25MHz `jtag_core_clk` inputs of the Smart VID controller.  
*Note:* Do not connect the `emif_usr_clk` signal to either the `vid_clk` or `jtag_core_clk` inputs. Doing so would hold both the EMIF IP and the SmartVID controller in a perpetual reset condition.
3. Instantiate the Arria 10 EMIF IP.
4. Connect the `vid_cal_done_persist` signal from the EMIF IP with the `cal_done_persistent` signal on the SmartVID controller. This connection enables handshaking between the EMIF IP and the SmartVID controller, which allows the EMIF IP to delay memory calibration until after voltage levels are stabilized.

*Note:* The EMIF `vid_cal_done_persist` interface becomes available only when a VID-enabled device is selected.

#### Related Information

[SmartVID Controller IP Core User Guide](#)

## 3.12. Hard Memory Controller Rate Conversion Feature

The hard memory controller's rate conversion feature allows the hard memory controller and PHY to run at half-rate, even though user logic is configured to run at quarter-rate.

To facilitate timing closure, you may choose to clock your core user logic at quarter-rate, resulting in easier timing closure at the expense of increased area and latency. To improve efficiency and help reduce overall latency, you can run the hard memory controller and PHY at half rate.

The rate conversion feature converts traffic from the FPGA core to the hard memory controller from quarter-rate to half-rate, and traffic from the hard memory controller to the FPGA core from half-rate to quarter-rate. From the perspective of user logic inside the FPGA core, the effect is the same as if the hard memory controller were running at quarter-rate.

The rate conversion feature is enabled automatically during IP generation whenever all of the following conditions are met:

- The hard memory controller is in use.
- User logic runs at quarter-rate.
- The interface targets either an ES2 or production device.
- Running the hard memory controller at half-rate does not exceed the fMax specification of the hard memory controller and hard PHY.

When the rate conversion feature is enabled, you should see the following info message displayed in the IP generation GUI:

PHY and controller running at 2x the frequency of user logic for improved efficiency.

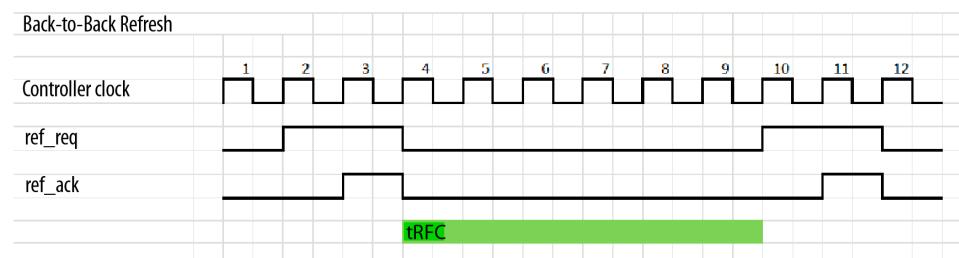
#### Related Information

[Arria 10 Core Fabric and General Purpose I/Os Handbook](#)

### 3.13. Back-to-Back User-Controlled Refresh for Hard Memory Controller

The following waveform illustrates the recommended Arria 10 model for back-to-back user-controlled refreshes, for optimized hard memory controller performance.

**Figure 103.**



You should deassert the refresh request after the refresh acknowledgement pulse is received. You can implement a timer to keep track of the tRFC status before asserting a refresh request. Failure to deassert the Refresh request can delay access to the rank not in refresh.

### 3.14. Compiling Arria 10 EMIF IP with the Quartus Prime Software



### 3.14.1. Instantiating the Arria 10 EMIF IP

Depending on your work flow, you may instantiate your IP with Qsys or with the IP Catalog.

#### Instantiating with Qsys

If you instantiate your IP as part of a system in Qsys, follow the Qsys documentation for information on instantiating the IP in a Quartus Prime project.

#### Instantiating with the IP Catalog

If you generated your IP with the IP Catalog, you must add the Quartus Prime IP file (.qip) to your Quartus Prime project. The .qip file identifies the names and locations of the files that compose the IP. After you add the .qip file to your project, you can instantiate the memory interface in the RTL.

### 3.14.2. Setting I/O Assignments in Arria 10 EMIF IP

The .qip file contains the I/O standard and I/O termination assignments required by the memory interface pins for proper operation. The assignment values are based on input that you provide during generation.

Unlike earlier device families, for Arria 10 EMIF IP you do not need to run a `<instance_name>_pin_assignments.tcl` script to add the assignments into the Quartus Prime Settings File (.qsf). The system reads and applies the assignments from the .qip file during every compilation, regardless of how you name the memory interface pins in the top-level design component. No new assignments are created in the project's .qsf file during compilation.

Note that I/O assignments in the .qsf file must specify the names of your top-level pins as target (-to), and you must not include the -entity or -library options.

Consult the generated .qip file for the set of I/O assignments that are provided with the IP.

#### Changing I/O Assignments

You should not make changes to the generated .qip file, because any changes are overwritten and lost when you regenerate the IP. If you want to override an assignment made in the .qip file, add the desired assignment to the project's .qsf file. Assignments in the .qsf file always take precedence over assignments in the .qip file.

## 3.15. Debugging Arria 10 EMIF IP

You can debug hardware failures by connecting to the EMIF Debug Toolkit or by exporting an Avalon-MM slave port, from which you can access information gathered during calibration. You can also connect to this port to mask ranks and to request recalibration.

You can access the exported Avalon-MM port in two ways:

- Via the External Memory Interface Debug Toolkit
- Via On-Chip Debug (core logic on the FPGA)



### 3.15.1. External Memory Interface Debug Toolkit

The External Memory Interface Debug Toolkit provides access to data collected by the Nios II sequencer during memory calibration, and allows you to perform certain tasks.

The External Memory Interface Debug Toolkit provides access to data including the following:

- General interface information, such as protocol and interface width
- Calibration results per group, including pass/fail status, failure stage, and delay settings

You can also perform the following tasks:

- Mask ranks from calibration (you might do this to skip specific ranks)
- Request recalibration of the interface

### 3.15.2. On-Chip Debug for Arria 10

The On-Chip Debug feature allows user logic to access the same debug capabilities as the External Memory Interface Toolkit. You can use On-Chip Debug to monitor the calibration results of an external memory interface, without a connected computer.

To use On-Chip Debug, you need a C header file which is provided as part of the external memory interface IP. The C header file defines data structures that contain calibration data, and definitions of the commands that can be sent to the memory interface.

The On-Chip Debug feature accesses the data structures through the Avalon-MM port that is exposed by the EMIF IP when you turn on debugging features.

### 3.15.3. Configuring Your EMIF IP for Use with the Debug Toolkit

The Arria 10 EMIF Debug Interface IP core contains the access point through which the EMIF Debug Toolkit reads calibration data collected by the Nios II sequencer.

#### Connecting an EMIF IP Core to an Arria 10 EMIF Debug Interface

For the EMIF Debug Toolkit to access the calibration data for an Arria 10 EMIF IP core, you must connect one of the EMIF cores in each I/O column to an Arria 10 EMIF Debug Interface IP core. Subsequent EMIF IP cores in the same column must connect in a daisy chain to the first.



There are two ways that you can add the Arria 10 EMIF Debug Interface IP core to your design:

- When you generate your EMIF IP core, on the **Diagnostics** tab, select **Add EMIF Debug Interface** for the **EMIF Debug Toolkit/On-Chip Debug Port**; you do not have to separately instantiate an Arria 10 EMIF Debug Interface core. This method does not export an Avalon-MM slave port. You can use this method if you require only EMIF Debug Toolkit access to this I/O column; that is, if you do not require On-Chip Debug Port access, or PHYLite reconfiguration access.
- When you generate your EMIF IP core, on the **Diagnostics** tab, select **Export** for the **EMIF Debug Toolkit/On-Chip Debug Port**. Then, separately instantiate an Arria 10 EMIF Debug Interface core and connect its `to_ioaux` interface to the `cal_debug` interface on the EMIF IP core. This method is appropriate if you want to also have On-Chip Debug Port access to this I/O column, or PHYLite reconfiguration access.

For each of the above methods, you must assign a unique interface ID for each external memory interface in the I/O column, to identify that interface in the Debug Toolkit. You can assign an interface ID using the dropdown list that appears when you enable the **Debug Toolkit/On-Chip Debug Port** option.

#### Daisy-Chaining Additional EMIF IP Cores for Debugging

After you have connected an Arria 10 EMIF Debug Interface to one of the EMIF IP cores in an I/O column, you must then connect subsequent EMIF IP cores in that column in a daisy-chain manner. If you don't require debug capabilities for a particular EMIF IP core, you do not have to connect that core to the daisy chain.

To create a daisy chain of EMIF IP cores, follow these steps:

1. On the first EMIF IP core, select **Enable Daisy-Chaining for EMIF Debug Toolkit/On-Chip Debug Port** to create an Avalon-MM interface called `cal_debug_out`.
2. On the second EMIF IP core, select **Export** as the **EMIF Debug Toolkit/On-Chip Debug Port** mode, to export an Avalon-MM interface called `cal_debug`.
3. Connect the `cal_debug_out` interface of the first EMIF core to the `cal_debug` interface of the second EMIF core.
4. To connect more EMIF cores to the daisy chain, select the **Enable Daisy-Chaining for EMIF Debug Toolkit/On-Chip Debug Port** option on the second core, connect it to the next core using the **Export** option as described above. Repeat the process for subsequent EMIF cores.

If you place any PHYLite cores with dynamic reconfiguration enabled into the same I/O column as an EMIF IP core, you should instantiate and connect the PHYLite cores in a similar way. See the *Altera PHYLite for Memory Megafunction User Guide* for more information.

#### 3.15.4. Arria 10 EMIF Debugging Examples

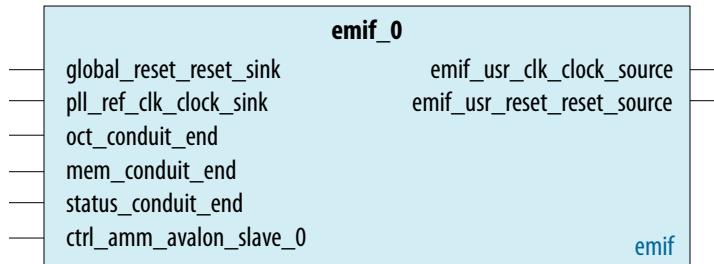
This topic provides examples of debugging a single external memory interface, and of adding additional EMIF instances to an I/O column.

### Debugging a Single External Memory Interface

- Under **EMIF Debug Toolkit/On-Chip Debug Port**, select **Add EMIF Debug Interface**.

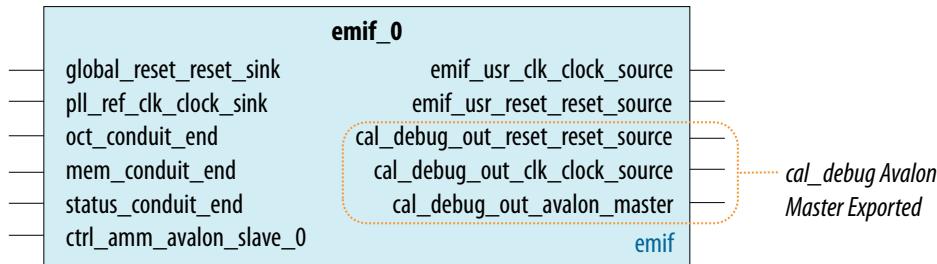
(If you want to use the On-Chip Debug Port instead of the EMIF Debug Toolkit, select **Export** instead.)

**Figure 104. EMIF With Debug Interface Added (No Additional Ports)**



- If you want to connect additional EMIF or PHYLite components in this I/O column, select *Enable Daisy Chaining* for *EMIF Debug Toolkit/On-Chip Debug Port*.

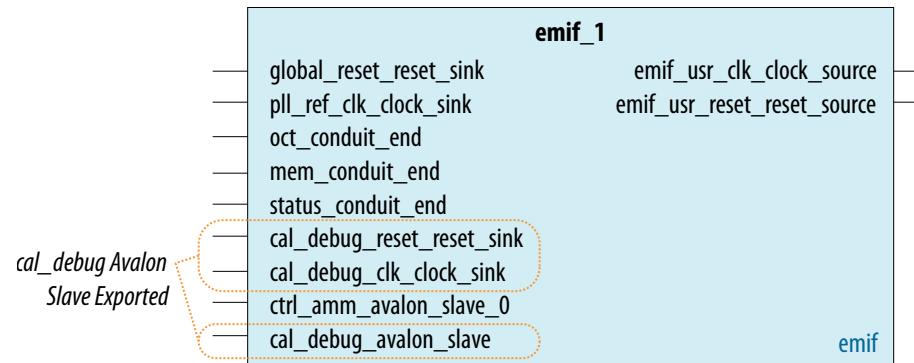
**Figure 105. EMIF With cal\_debug Avalon Master Exported**



### Adding Additional EMIF Instances to an I/O Column

- Under **EMIF Debug Toolkit/On-Chip Debug Port**, select **Export**.

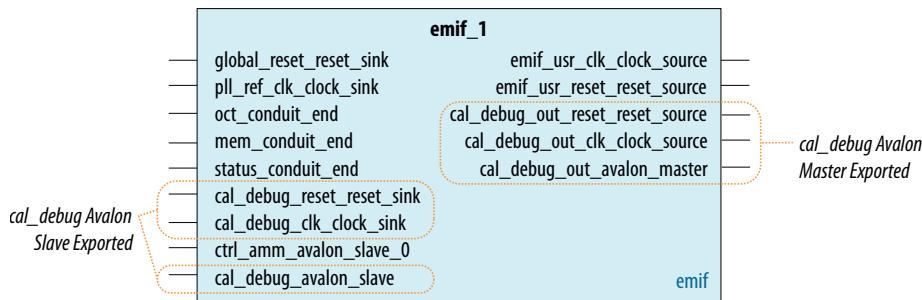
**Figure 106. EMIF With cal\_debug Avalon Slave Exported**



- Specify a unique interface ID for this EMIF instance.
- If you want to connect additional EMIF or PHYLite components in this I/O column, select *Enable Daisy Chaining* for *EMIF Debug Toolkit/On-Chip Debug Port*.

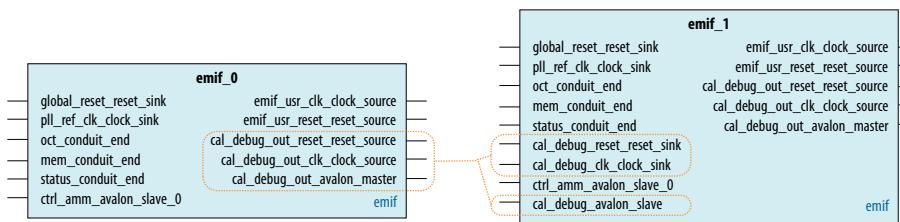


**Figure 107. EMIF With Both cal\_debug Master and Slave Exported**



4. Connect the cal\_debug Avalon Master, clock, and reset interfaces of the previous component to the cal\_debug Avalon Slave, clock, and reset interfaces of this component.

**Figure 108. EMIF Components Connected**



### 3.16. Arria 10 EMIF for Hard Processor Subsystem

The Arria 10 EMIF IP can enable the Arria 10 Hard Processor Subsystem (HPS) to access external DRAM memory devices.

To enable connectivity between the Arria 10 HPS and the Arria 10 EMIF IP, you must create and configure an instance of the Arria 10 External Memory Interface for HPS IP core, and use Qsys to connect it to the Arria 10 Hard Processor Subsystem instance in your system.

#### Supported Modes

The Arria 10 Hard Processor Subsystem is compatible with the following external memory configurations:

|   |                                      |
|---|--------------------------------------|
| Protocol  | DDR3, DDR4                           |
| Maximum memory clock frequency                  | DDR3: 1.067 GHz<br>DDR4: 1.333 GHz   |
| Configuration                                   | Hard PHY with hard memory controller |
| Clock rate of PHY and hard memory controller    | Half-rate                            |
| Data width (without ECC)                        | 16-bit, 32-bit, 64-bit <sup>2</sup>  |
| Data width (with ECC)                           | 24-bit, 40-bit, 72-bit <sup>2</sup>  |
| DQ width per group                              | x8                                   |
| Maximum number of I/O lanes for address/command | 3                                    |

*continued...*



|  |                                |
|--|--------------------------------|
| Memory format  | Discrete, UDIMM, SODIMM, RDIMM |
| Ranks / CS# width  | Up to 2                        |
| Notes to table:  |                                |
| 1. Only Arria 10 devices with a special ordering code support 64-bit and 72-bit data widths; all other devices support only to 32-bit data widths. |                                |

*Note:* Arria 10 HPS EMIF IP does not currently support SmartVID.

### 3.16.1. Restrictions on I/O Bank Usage for Arria 10 EMIF IP with HPS

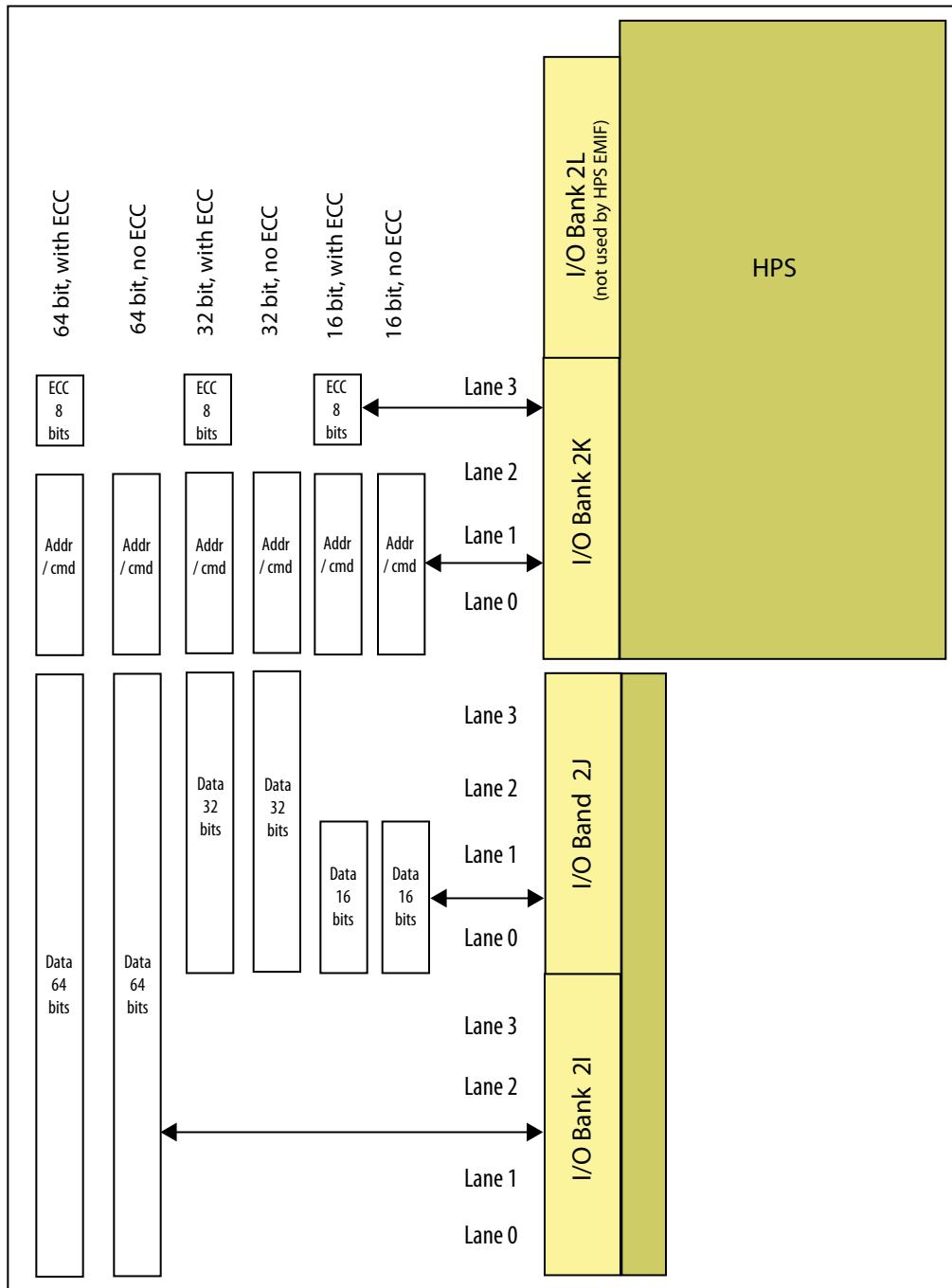
Only certain Arria 10 I/O banks can be used to implement Arria 10 EMIF IP with the Arria 10 Hard Processor Subsystem. If both Arria 10 HPS EMIF IP and non-HPS Arria 10 EMIF IP are implemented, you must place the non-HPS EMIF IP in a different I/O column than the HPS EMIF IP.

The restrictions on I/O bank usage result from the Arria 10 HPS having hard-wired connections to the EMIF circuits in the I/O banks closest to the HPS. For any given EMIF configuration, the pin-out of the EMIF-to-HPS interface is fixed.



The following diagram illustrates the use of I/O banks and lanes for various EMIF-HPS data widths:

**Figure 109. I/O Banks and Lanes Usage**



You should refer to the pinout file for your device and package for detailed information. Banks and pins used for HPS access to a DDR interface are labeled HPS\_DDR in the HPS Function column of the pinout file.



By default, the Arria 10 External Memory Interface for HPS IP core together with the Quartus Prime Fitter automatically implement the correct pin-out for HPS EMIF without you having to implement additional constraints. If, for any reason, you must modify the default pin-out, you must adhere to the following requirements, which are specific to HPS EMIF:

1. Within a single data lane (which implements a single x8 DQS group):
  - a. DQ pins must use pins at indices 1, 2, 3, 6, 7, 8, 9, 10. You may swap the locations between the DQ bits (that is, you may swap location of DQ[0] and DQ[3]) so long as the resulting pin-out uses pins at these indices only.
  - b. DM/DBI pin must use pin at index 11. There is no flexibility.
  - c. DQS/DQS# must use pins at index 4 and 5. There is no flexibility.
2. Assignment of data lanes must be as illustrated in the above figure. You are allowed to swap the locations of entire byte lanes (that is, you may swap locations of byte 0 and byte 3) so long as the resulting pin-out uses only the lanes permitted by your HPS EMIF configuration, as shown in the above figure.
3. You must not change placement of the address and command pins from the default.
4. You may place the alert# pin at any available pin location in either a data lane or an address and command lane.

To override the default generated pin assignments, comment out the relevant HPS\_LOCATION assignments in the .qip file, and add your own location assignments (using set\_location\_assignment) in the .qsf file.

### 3.16.2. Using the EMIF Debug Toolkit with Arria 10 HPS Interfaces

The External Memory Interface Debug Toolkit is not directly compatible with Arria 10 HPS interfaces.

To debug your Arria 10 HPS interface using the EMIF Debug Toolkit, you should create an identically parameterized, non-HPS version of your interface, and apply the EMIF Debug Toolkit to that interface. When you finish debugging this non-HPS interface, you can then apply any needed changes to your HPS interface, and continue your design development.

## 3.17. Arria 10 EMIF Ping Pong PHY

Ping Pong PHY allows two memory interfaces to share the address and command bus through time multiplexing. Compared to having two independent interfaces that allocate address and command lanes separately, Ping Pong PHY achieves the same throughput with fewer resources, by sharing the address and command lanes.

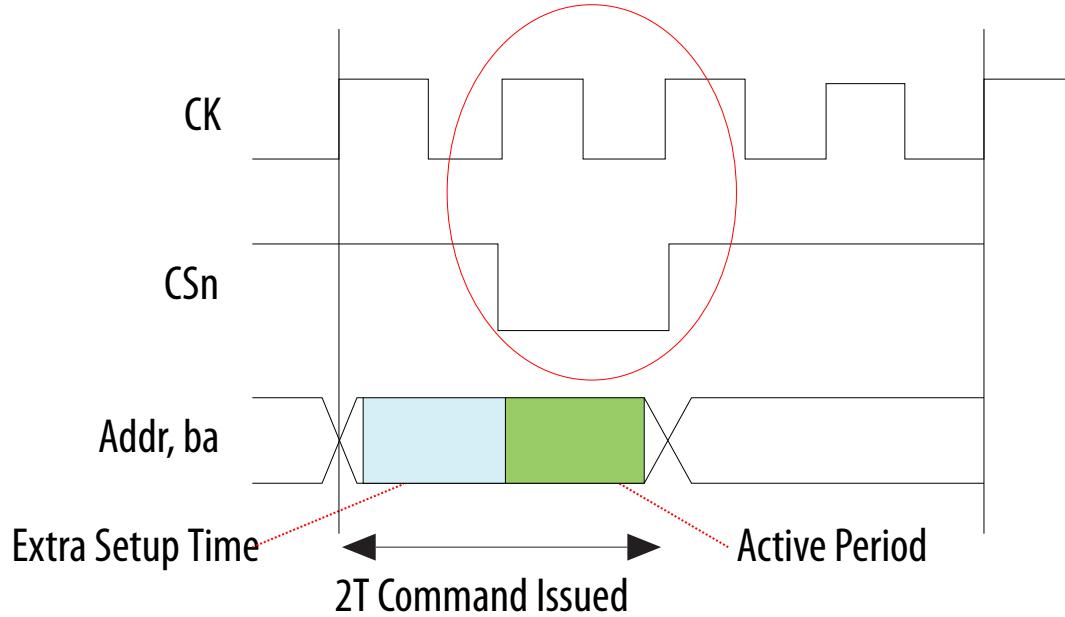
In Arria 10 EMIF, Ping Pong PHY supports both half-rate and quarter-rate interfaces for DDR3, and quarter-rate for DDR4.

### 3.17.1. Ping Pong PHY Feature Description

Conventionally, the address and command buses of a DDR3 or DDR4 half-rate or quarter-rate interface use 2T time—meaning that commands are issued for two full-rate clock cycles, as illustrated below.

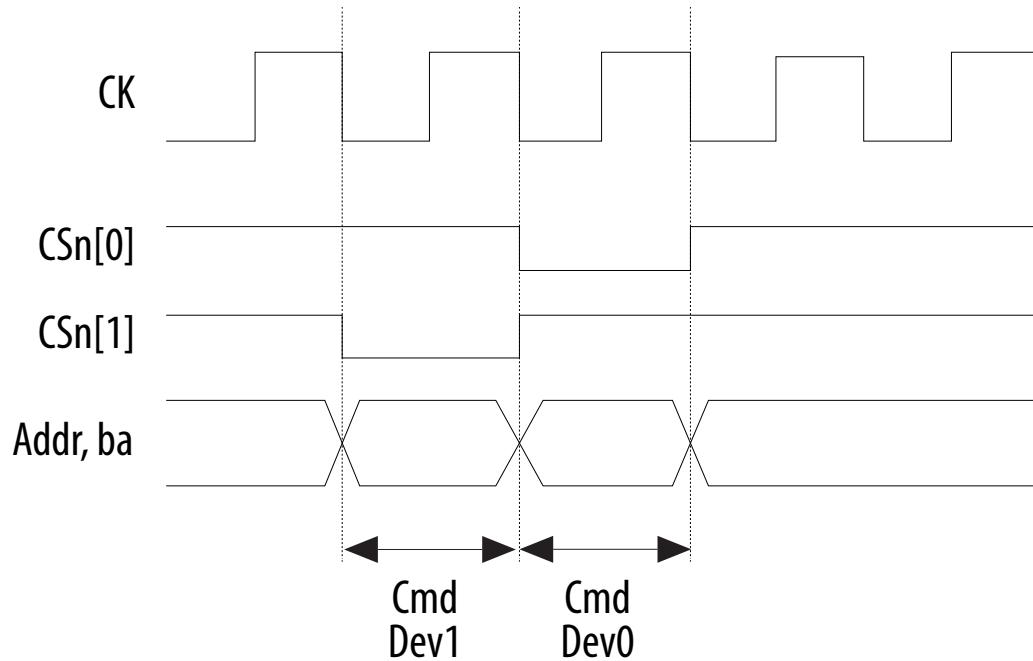


**Figure 110. 2T Command Timing**



With the Ping Pong PHY, address and command signals from two independent controllers are multiplexed onto shared buses by delaying one of the controller outputs by one full-rate clock cycle. The result is 1T timing, with a new command being issued on each full-rate clock cycle. The following figure shows address and command timing for the Ping Pong PHY.

**Figure 111. 1T Command Timing Use by Ping Pong PHY**



### 3.17.2. Ping Pong PHY Architecture

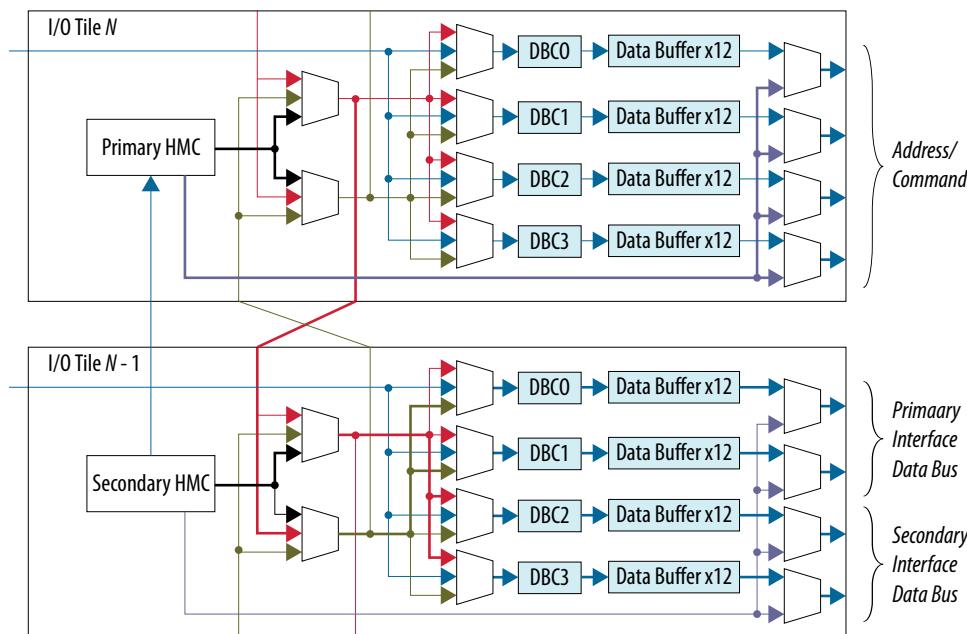
In Arria 10 EMIF, the Ping Pong PHY feature can be enabled only with the hard memory controller, where two hard memory controllers are instantiated—one for the primary interface and one for the secondary interface.

The hard memory controller I/O bank of the primary interface is used for address and command and is always adjacent and above the hard memory controller bank of the secondary interface. All four lanes of the primary hard memory controller bank are used for address and command. The I/O bank containing the secondary hard memory controller must have at least one lane from the secondary interface.

The following example shows a 2x16 Ping Pong PHY bank-lane configuration. The upper bank (I/O bank N) is the address and command bank, which serves both the primary and secondary interfaces. The primary hard memory controller is linked to the secondary interface by the Ping Pong bus. The lower bank (I/O bank N-1) is the secondary interface bank, which carries the data buses for both primary and secondary interfaces. In the 2x16 case a total of four I/O banks are required for data, hence two banks in total are sufficient for the implementation.

The data for the primary interface is routed down to the top two lanes of the secondary I/O bank, and the data for the secondary interface is routed to the bottom two lanes of the secondary I/O bank.

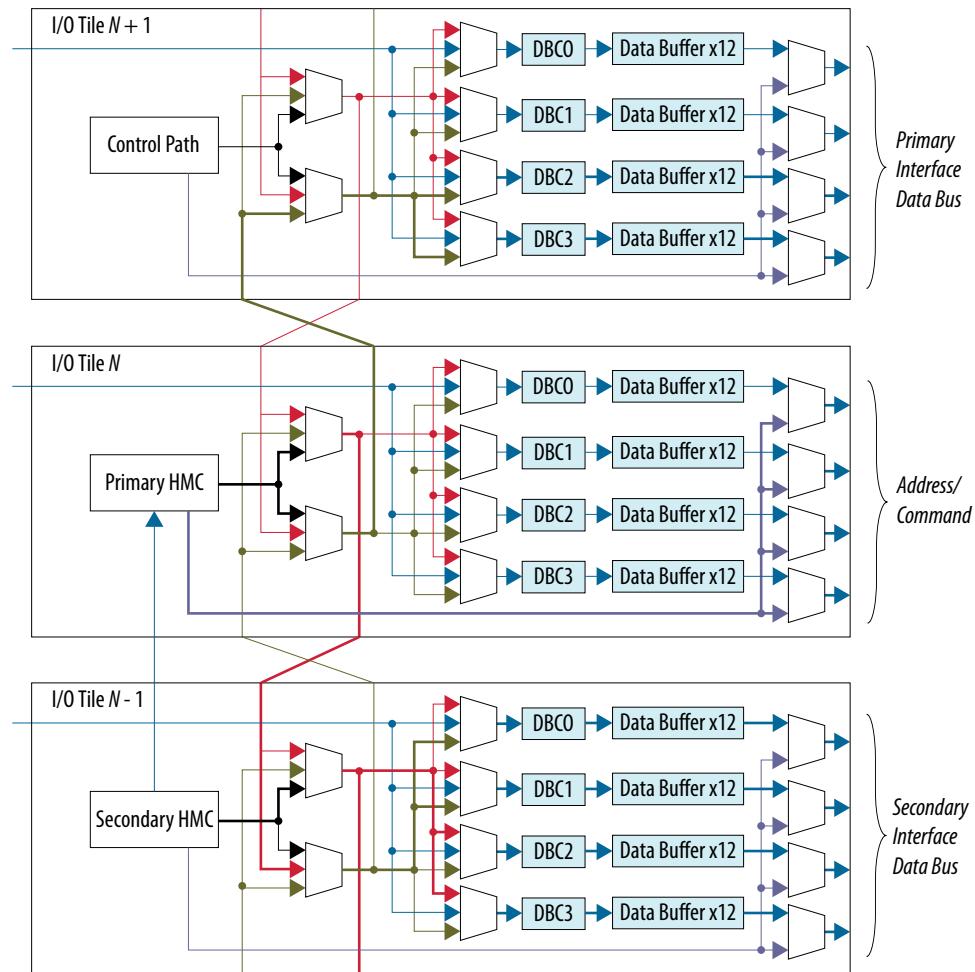
**Figure 112. 2x16 Ping Pong PHY I/O Bank-Lane Configuration**



A 2x32 interface can be implemented using three tiles, so long as the tile containing the secondary hard memory controller has at least one secondary data lane. The order of the lanes does not matter.



**Figure 113. 2x32 Ping Pong PHY I/O Bank-Lane Configuration.**



### 3.17.3. Ping Pong PHY Limitations

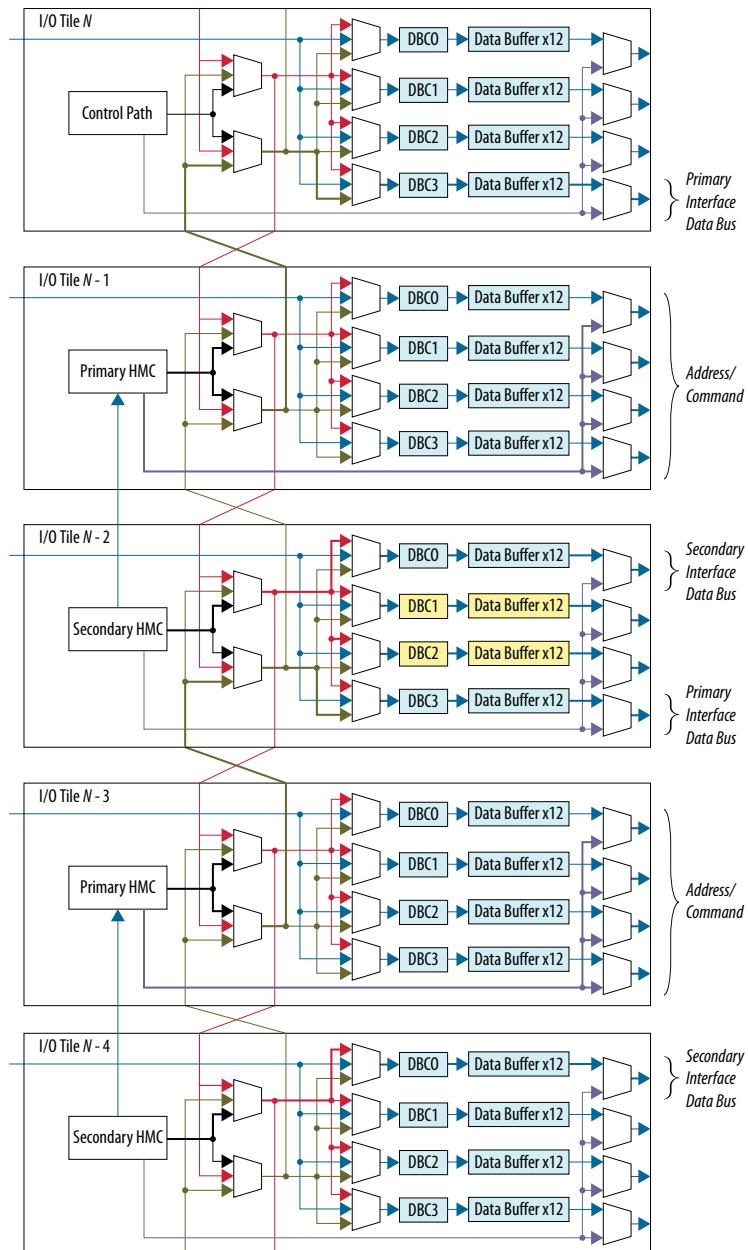
Ping Pong PHY supports up to two ranks per memory interface. In addition, the maximum data width is x72, which is half the maximum width of x144 for a single interface.

Ping Pong PHY uses all lanes of the address and command I/O bank as address and command. For information on the pin allocations of the DDR3 and DDR4 address and command I/O bank, refer to *DDR3 Scheme 1* and *DDR4 Scheme 3*, in *External Memory Interface Pin Information for Arria 10 Devices*, on [www.altera.com](http://www.altera.com).

An additional limitation is that I/O lanes may be left unused when you instantiate multiple pairs of Ping Pong PHY interfaces. The following diagram shows two pairs of x8 Pin Pong controllers (a total of 4 interfaces). Lanes highlighted in yellow are not driven by any memory interfaces (unused lanes and pins can still serve as general purpose I/Os). Even with some I/O lanes left unused, the Ping Pong PHY approach is still beneficial in terms of resource usage, compared to independent interfaces.

Memory widths of 24 bits and 40 bits have a similar situation, while 16 bit, 32 bit, and 64 bit memory widths do not suffer this limitation.

**Figure 114. Two Pairs of x8 Pin-Pong PHY Controllers**



#### Related Information

[External Memory Interface Pin Information for Arria 10 Devices](#)

#### 3.17.4. Ping Pong PHY Calibration

A Ping Pong PHY interface is calibrated as a regular interface of double width.



Calibration of a Ping Pong PHY interface incorporates two sequencers, one on the primary hard memory controller I/O bank, and one on the secondary hard memory controller I/O bank. To ensure that the two sequencers issue instructions on the same memory clock cycle, the Nios II processor configures the sequencer on the primary hard memory controller to receive a token from the secondary interface, ignoring any commands from the Avalon bus. Additional delays are programmed on the secondary interface to allow for the passing of the token from the sequencer on the secondary hard memory controller tile to the sequencer on the primary hard memory controller tile. During calibration, the Nios II processor assumes that commands are always issued from the sequencer on the primary hard memory controller I/O bank. After calibration, the Nios II processor adjusts the delays for use with the primary and secondary hard memory controllers.

### 3.17.5. Using the Ping Pong PHY

The following steps describe how to use the Ping Pong PHY for Arria 10 EMIF.

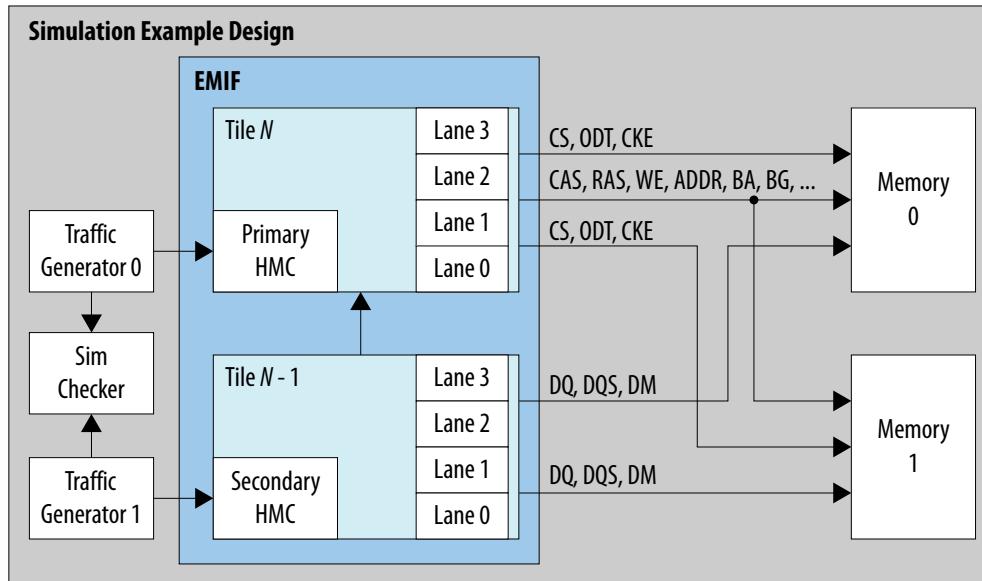
1. Configure a single memory interface according to your requirements.
2. Select **Instantiate two controllers sharing a Ping Pong PHY** on the **General** tab in the parameter editor.  
The Quartus Prime software replicates the interface, resulting in two memory controllers and a shared PHY. The system configures the I/O bank-lane structure, without further input from you.

### 3.17.6. Ping Pong PHY Simulation Example Design

The following figure illustrates a top-level block diagram of a generated Ping Pong PHY simulation example design, using two I/O banks.

Functionally, the IP interfaces with user traffic separately, as it would with two independent memory interfaces. You can also generate synthesizable example designs, where the external memory interface IP interfaces with a traffic generator.

**Figure 115. Ping Pong PHY Simulation Example Design**



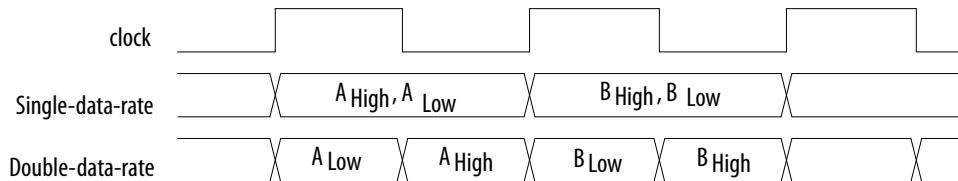
## 3.18. AFI 4.0 Specification

The Altera PHY interface (AFI) 4.0 defines communication between the controller and physical layer (PHY) in the external memory interface IP.

The AFI is a single-data-rate interface, meaning that data is transferred on the rising edge of each clock cycle. Most memory interfaces, however, operate at double-data-rate, transferring data on both the rising and falling edges of the clock signal. If the AFI interface is to directly control a double-data-rate signal, two single-data-rate bits must be transmitted on each clock cycle; the PHY then sends out one bit on the rising edge of the clock and one bit on the falling edge.

The AFI convention is to send the low part of the data first and the high part second, as shown in the following figure.

**Figure 116. Single Versus Double Data Rate Transfer**



### 3.18.1. Bus Width and AFI Ratio

In cases where the AFI clock frequency is one-half or one-quarter of the memory clock frequency, the AFI data must be twice or four times as wide, respectively, as the corresponding memory data. The ratio between AFI clock and memory clock frequencies is referred to as the AFI ratio. (A half-rate AFI interface has an AFI ratio of 2, while a quarter-rate interface has an AFI ratio of 4.)

In general, the width of the AFI signal depends on the following three factors:

- The size of the equivalent signal on the memory interface. For example, if `a[15:0]` is a DDR3 address input and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_addr` bus will be 16-bits wide.
- The data rate of the equivalent signal on the memory interface. For example, if `d[7:0]` is a double-data-rate QDR II input data bus and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_write_data` bus will be 16-bits wide.
- The AFI ratio. For example, if `cs_n` is a single-bit DDR3 chip select input and the AFI clock runs at half the speed of the memory interface, the equivalent `afi_cs_n` bus will be 2-bits wide.

The following formula summarizes the three factors described above:

```
AFI_width = memory_width * signal_rate * AFI_RATE_RATIO
```

**Note:** The above formula is a general rule, but not all signals obey it. For definite signal-size information, refer to the specific table.



### 3.18.2. AFI Parameters

The following tables list Altera PHY interface (AFI) parameters for AFI 4.0. Not all parameters are used for all protocols.

The parameters described in the following tables affect the width of AFI signal buses. Parameters prefixed by MEM\_IF\_ refer to the signal size at the interface between the PHY and memory device.

**Table 46. Ratio Parameters**

| Parameter Name  | Description  |
|-----------------|--|
| AFI_RATE_RATIO  | The ratio between the AFI clock frequency and the memory clock frequency. For full-rate interfaces this value is 1, for half-rate interfaces the value is 2, and for quarter-rate interfaces the value is 4. |
| DATA_RATE_RATIO | The number of data bits transmitted per clock cycle. For single-data rate protocols this value is 1, and for double-data rate protocols this value is 2.   |
| ADDR_RATE_RATIO | The number of address bits transmitted per clock cycle. For single-data rate address protocols this value is 1, and for double-data rate address protocols this value is 2.                                  |

**Table 47. Memory Interface Parameters**

| Parameter Name         | Description  |
|------------------------|--|
| MEM_IF_ADDR_WIDTH      | The width of the address bus on the memory device(s). For LPDDR3, the width of the CA bus, which encodes commands and addresses together.                  |
| MEM_IF_BANKGROUP_WIDTH | The width of the bank group bus on the interface to the memory device(s). Applicable to DDR4 only.   |
| MEM_IF_BANKADDR_WIDTH  | The width of the bank address bus on the interface to the memory device(s). Typically, the log 2 of the number of banks. Not applicable to DDR4.           |
| MEM_IF_CS_WIDTH        | The number of chip selects on the interface to the memory device(s).   |
| MEM_IF_CKE_WIDTH       | Number of CKE signals on the interface to the memory device(s). This usually equals to MEM_IF_CS_WIDTH except for certain DIMM configurations.             |
| MEM_IF_ODT_WIDTH       | Number of ODT signals on the interface to the memory device(s). This usually equals to MEM_IF_CS_WIDTH except for certain DIMM configurations.             |
| MEM_IF_WRITE_DQS_WIDTH | The number of DQS (or write clock) signals on the write interface. For example, the number of DQS groups.  |
| MEM_IF_CLK_PAIR_COUNT  | The number of CK/CK# pairs.  |
| MEM_IF_DQ_WIDTH        | The number of DQ signals on the interface to the memory device(s). For single-ended interfaces such as QDR II, this value is the number of D or Q signals. |
| MEM_IF_DM_WIDTH        | The number of data mask pins on the interface to the memory device(s).   |

**Table 48. Derived AFI Parameters**

| Parameter Name      | Derivation Equation  |
|---------------------|--|
| AFI_ADDR_WIDTH      | MEM_IF_ADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO                                   |
| AFI_BANKGROUP_WIDTH | MEM_IF_BANKGROUP_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO.<br>Applicable to DDR4 only. |
| AFI_BANKADDR_WIDTH  | MEM_IF_BANKADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO                               |
| <i>continued...</i> |  |



| Parameter Name      | Derivation Equation                                       |
|---------------------|---|
| AFI_CONTROL_WIDTH   | AFI_RATE_RATIO * ADDR_RATE_RATIO                          |
| AFI_CS_WIDTH        | MEM_IF_CS_WIDTH * AFI_RATE_RATIO                          |
| AFI_CKE_WIDTH       | MEM_IF_CKE_WIDTH * AFI_RATE_RATIO                         |
| AFI_ODT_WIDTH       | MEM_IF_ODT_WIDTH * AFI_RATE_RATIO                         |
| AFI_DM_WIDTH        | MEM_IF_DM_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO        |
| AFI_DQ_WIDTH        | MEM_IF_DQ_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO        |
| AFI_WRITE_DQS_WIDTH | MEM_IF_WRITE_DQS_WIDTH * AFI_RATE_RATIO                   |
| AFI_LAT_WIDTH       | 6   |
| AFI_RLAT_WIDTH      | AFI_LAT_WIDTH   |
| AFI_WLAT_WIDTH      | AFI_LAT_WIDTH * MEM_IF_WRITE_DQS_WIDTH                    |
| AFI_CLK_PAIR_COUNT  | MEM_IF_CLK_PAIR_COUNT                                     |
| AFI_WRANK_WIDTH     | Number of ranks * MEM_IF_WRITE_DQS_WIDTH * AFI_RATE_RATIO |
| AFI_RRANK_WIDTH     | Number of ranks * MEM_IF_READ_DQS_WIDTH * AFI_RATE_RATIO  |

### 3.18.3. AFI Signals

The following tables list Altera PHY interface (AFI) signals grouped according to their functions.

In each table, the **Direction** column denotes the direction of the signal relative to the PHY. For example, a signal defined as an output passes out of the PHY to the controller. The AFI specification does not include any bidirectional signals.

Not all signals are used for all protocols.

#### 3.18.3.1. AFI Clock and Reset Signals

The AFI interface provides up to two clock signals and an asynchronous reset signal.

**Table 49. Clock and Reset Signals**

| Signal Name  | Direction | Width | Description  |
|--------------|-----------|-------|--|
| afi_clk      | Output    | 1     | Clock with which all data exchanged on the AFI bus is synchronized. In general, this clock is referred to as full-rate, half-rate, or quarter-rate, depending on the ratio between the frequency of this clock and the frequency of the memory device clock. |
| afi_half_clk | Output    | 1     | Clock signal that runs at half the speed of the afi_clk. The controller uses this signal when the half-rate bridge feature is in use. This signal is optional.   |
| afi_reset_n  | Output    | 1     | Asynchronous reset output signal. You must synchronize this signal to the clock domain in which you use it.  |



### 3.18.3.2. AFI Address and Command Signals

The address and command signals for AFI 4.0 encode read/write/configuration commands to send to the memory device. The address and command signals are single-data rate signals.

**Table 50. Address and Command Signals**

| Signal Name | Direction | Width                 | Description   |
|-------------|-----------|-----------------------|---|
| afi_addr    | Input     | AFI_ADDR_WIDTH        | Address or CA bus (LPDDR3 only). ADDR_RATE_RATIO is 2 for LPDDR3 CA bus.  |
| afi_bg      | Input     | AFI_BANKGROUP_WIDTH   | Bank group (DDR4 only).   |
| afi_ba      | Input     | AFI_BANKADDR_WIDTH    | Bank address. (Not applicable for LPDDR3.)  |
| afi_cke     | Input     | AFI_CLK_EN_WIDTH      | Clock enable.   |
| afi_cs_n    | Input     | AFI_CS_WIDTH          | Chip select signal. (The number of chip selects may not match the number of ranks; for example, RDIMMs and LRDIMMs require a minimum of 2 chip select signals for both single-rank and dual-rank configurations. Consult your memory device data sheet for information about chip select signal width.) (Matches the number of ranks for LPDDR3.) |
| afi_ras_n   | Input     | AFI_CONTROL_WIDTH     | RAS# (for DDR2 and DDR3 memory devices.)  |
| afi_we_n    | Input     | AFI_CONTROL_WIDTH     | WE# (for DDR2, DDR3, and RLDRAM II memory devices.)   |
| afi_rw_n    | Input     | AFI_CONTROL_WIDTH * 2 | RWA/B# (QDR-IV).  |
| afi_cas_n   | Input     | AFI_CONTROL_WIDTH     | CAS# (for DDR2 and DDR3 memory devices.)  |
| afi_act_n   | Input     | AFI_CONTROL_WIDTH     | ACT# (DDR4).  |
| afi_ref_n   | Input     | AFI_CONTROL_WIDTH     | REF# (for RLDRAM II memory devices.)  |
| afi_RST_n   | Input     | AFI_CONTROL_WIDTH     | RESET# (for DDR3 and DDR4 memory devices.)  |
| afi_odi     | Input     | AFI_CLK_EN_WIDTH      | On-die termination signal for DDR2, DDR3, and LPDDR3 memory devices. (Do not confuse this memory device signal with the FPGA's internal on-chip termination signal.)  |
| afi_par     | Input     | AFI_CS_WIDTH          | Address and command parity input. (DDR4)<br>Address parity input. (QDR-IV)  |
| afi_ainv    | Input     | AFI_CONTROL_WIDTH     | Address inversion. (QDR-IV)   |

*continued...*



| Signal Name         | Direction | Width              | Description  |
|---------------------|-----------|--------------------|--|
| afi_mem_clk_disable | Input     | AFI_CLK_PAIR_COUNT | When this signal is asserted, mem_clk and mem_clk_n are disabled. This signal is used in low-power mode. |
| afi_wps_n           | Output    | AFI_CS_WIDTH       | WPS (for QDR II/II+ memory devices.)   |
| afi_rps_n           | Output    | AFI_CS_WIDTH       | RPS (for QDR II/II+ memory devices.)   |

### 3.18.3.3. AFI Write Data Signals

Write Data Signals for AFI 4.0 control the data, data mask, and strobe signals passed to the memory device during write operations.

**Table 51. Write Data Signals**

| Signal Name     | Direction | Width                   | Description  |
|-----------------|-----------|-------------------------|--|
| afi_dqs_burst   | Input     | AFI_RATE_RATIO          | Controls the enable on the strobe (DQS) pins for DDR2, DDR3, LPDDR2, and LPDDR3 memory devices. When this signal is asserted, mem_dqs and mem_dqsn are driven.<br>This signal must be asserted before afi_wdata_valid to implement the write preamble, and must be driven for the correct duration to generate a correctly timed mem_dqs signal. |
| afi_wdata_valid | Input     | AFI_RATE_RATIO          | Write data valid signal. This signal controls the output enable on the data and data mask pins.  |
| afi_wdata       | Input     | AFI_DQ_WIDTH            | Write data signal to send to the memory device at double-data rate. This signal controls the PHY's mem_dq output.  |
| afi_dm          | Input     | AFI_DM_WIDTH            | Data mask. This signal controls the PHY's mem_dm signal for DDR2, DDR3, LPDDR2, LPDDR3, and RLDRAM II memory devices.)<br>Also directly controls the PHY's mem_db signal for DDR4.<br>The mem_dm and mem_db features share the same port on the memory device.   |
| afi_bws_n       | Input     | AFI_DM_WIDTH            | Data mask. This signal controls the PHY's mem_bws_n signal for QDR II/II+ memory devices.  |
| afi_dinv        | Input     | AFI_WRITE_DQS_WIDTH * 2 | Data inversion. It directly controls the PHY's mem_dinva/b signal for QDR-IV devices.  |

### 3.18.3.4. AFI Read Data Signals

Read Data Signals for AFI 4.0 control the data sent from the memory device during read operations.



**Table 52. Read Data Signals**

| Signal Name       | Direction | Width          | Description  |
|-------------------|-----------|----------------|--|
| afi_rdata_en_full | Input     | AFI_RATE_RATIO | Read data enable full. Indicates that the memory controller is currently performing a read operation. This signal is held high for the entire read burst. If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2). |
| afi_rdata         | Output    | AFI_DQ_WIDTH   | Read data from the memory device. This data is considered valid only when afi_rdata_valid is asserted by the PHY.  |
| afi_rdata_valid   | Output    | AFI_RATE_RATIO | Read data valid. When asserted, this signal indicates that the afi_rdata bus is valid. If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2).  |

### 3.18.3.5. AFI Calibration Status Signals

The PHY instantiates a sequencer which calibrates the memory interface with the memory device and some internal components such as read FIFOs and valid FIFOs. The sequencer reports the results of the calibration process to the controller through the Calibration Status Signals in the AFI interface.

**Table 53. Calibration Status Signals**

| Signal Name  | Direction | Width          | Description   |
|--|-----------|----------------|---|
| afi_cal_success  | Output    | 1              | Asserted to indicate that calibration has completed successfully.   |
| afi_cal_fail   | Output    | 1              | Asserted to indicate that calibration has failed.   |
| afi_cal_req  | Input     | 1              | Effectively a synchronous reset for the sequencer. When this signal is asserted, the sequencer returns to the reset state; when this signal is released, a new calibration sequence begins.   |
| afi_wlat   | Output    | AFI_WLAT_WIDTH | The required write latency in afi_clk cycles, between address/command and write data being issued at the PHY/controller interface. The afi_wlat value can be different for different groups; each group's write latency can range from 0 to 63. If write latency is the same for all groups, only the lowest 6 bits are required. |
| afi_rlat<br><sup>(1)</sup>   | Output    | AFI_RLAT_WIDTH | The required read latency in afi_clk cycles between address/command and read data being returned to the PHY/controller interface. Values can range from 0 to 63.  |
| Note to Table:   |           |                |   |
| 1. The afi_rlat signal is not supported for PHY-only designs. Instead, you can sample the afi_rdata_valid signal to determine when valid read data is available. |           |                |   |



### 3.18.3.6. AFI Tracking Management Signals

When tracking management is enabled, the sequencer can take control over the AFI 4.0 interface at given intervals, and issue commands to the memory device to track the internal DQS Enable signal alignment to the DQS signal returning from the memory device. The tracking management portion of the AFI 4.0 interface provides a means for the sequencer and the controller to exchange handshake signals.

**Table 54. Tracking Management Signals**

| Signal Name          | Direction | Width | Description   |
|----------------------|-----------|-------|---|
| afi_ctl_refresh_done | Input     | 4     | Handshaking signal from controller to tracking manager, indicating that a refresh has occurred and waiting for a response.                                  |
| afi_seq_busy         | Output    | 4     | Handshaking signal from sequencer to controller, indicating when DQS tracking is in progress.   |
| afi_ctl_long_idle    | Input     | 4     | Handshaking signal from controller to tracking manager, indicating that it has exited low power state without a periodic refresh, and waiting for response. |

### 3.18.3.7. AFI Shadow Register Management Signals

Shadow registers are a feature that enables high-speed multi-rank support. Shadow registers allow the sequencer to calibrate each rank separately, and save the calibrated settings—such as deskew delay-chain configurations—of each rank in its own set of shadow registers.

During a rank-to-rank switch, the correct set of calibrated settings is restored just in time to optimize the data valid window. The PHY relies on additional AFI signals to control which set of shadow registers to activate.

**Table 55. Shadow Register Management Signals**

| Signal Name | Direction | Width           | Description   |
|-------------|-----------|-----------------|---|
| afi_wrank   | Input     | AFI_WRANK_WIDTH | Signal from controller specifying which rank the write data is going to. The signal timing is identical to that of afi_dqs_burst. That is, afi_wrank must be asserted at the same time and must last the same duration as the afi_dqs_burst signal.   |
| afi_rrank   | Output    | AFI_RRANK_WIDTH | Signal from controller specifying which rank is being read. The signal must be asserted at the same time as the afi_rdata_en signal when issuing a read command, but unlike afi_rdata_en, afi_rrank is stateful. That is, once asserted, the signal value must remain unchanged until the controller issues a new read command to a different rank. |



Both the afi\_wrank and afi\_rrank signals encode the rank being accessed using the one-hot scheme (e.g. in a quad-rank interface, 0001, 0010, 0100, 1000 refer to the 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> rank respectively). The ordering within the bus is the same as other AFI signals. Specifically the bus is ordered by time slots, for example:

Half-rate afi\_w/rrank = {T1, T0}

Quarter-rate afi\_w/rrank = {T3, T2, T1, T0}

Where Tx is a number of rank-bit words that one-hot encodes the rank being accessed at the y<sup>th</sup> full-rate cycle.

#### Additional Requirements for Arria 10 Shadow Register Support

To ensure that the hardware has enough time to switch from one shadow register to another, the controller must satisfy the following minimum rank-to-rank-switch delays (tRTRS):

- Two read commands going to different ranks must be separated by a minimum of 3 full-rate cycles (in addition to the burst length delay needed to avoid collision of data bursts).
- Two write commands going to different rank must be separated by a minimum of 4 full-rate cycles (in addition to the burst length delay needed to avoid collision of data bursts).

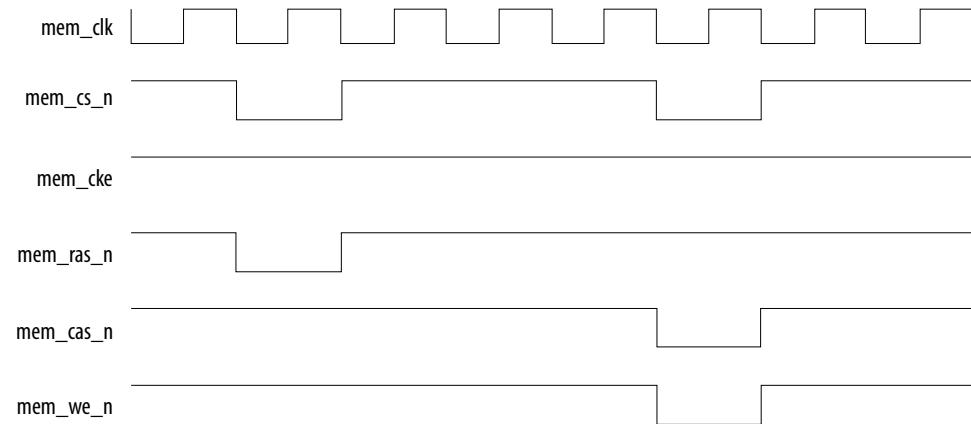
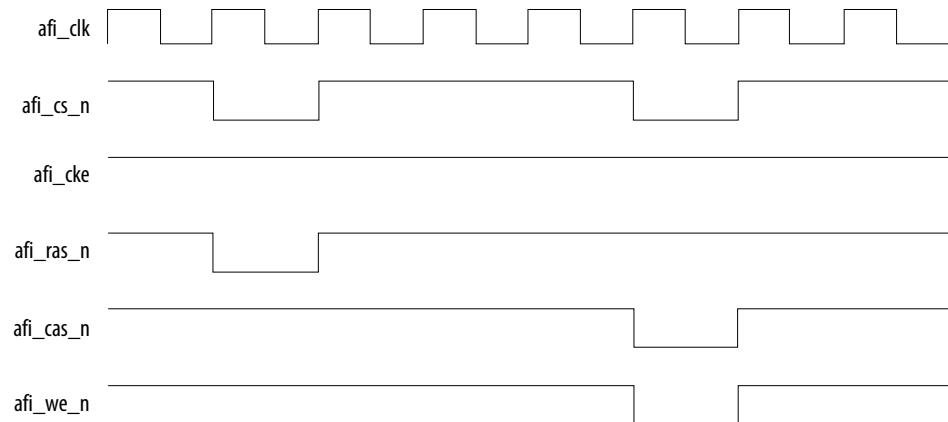
The Arria 10 device family supports a maximum of 4 sets of shadow registers, each for an independent set of timings. More than 4 ranks are supported if those ranks have four or fewer sets of independent timing. For example, the rank multiplication mode of an LRDIMM allows more than one physical rank to share a set of timing data as a single logical rank. Therefore Arria 10 devices can support up to 4 logical ranks, though that means more than 4 physical ranks.

### 3.18.4. AFI 4.0 Timing Diagrams

#### 3.18.4.1. AFI Address and Command Timing Diagrams

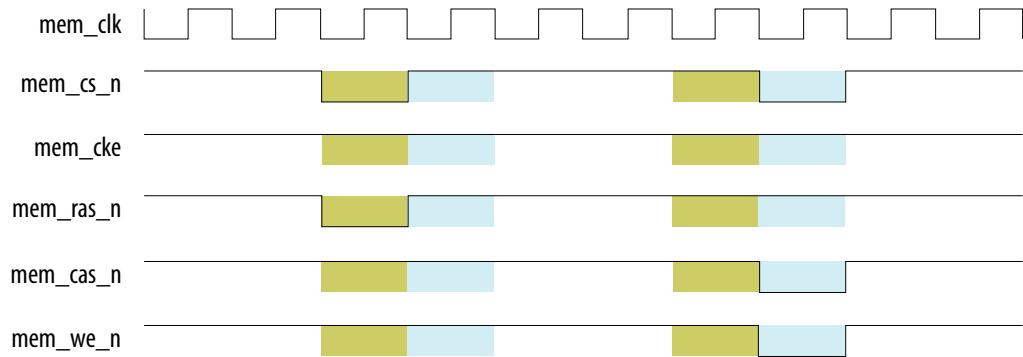
Depending on the ratio between the memory clock and the PHY clock, different numbers of bits must be provided per PHY clock on the AFI interface. The following figures illustrate the AFI address/command waveforms in full, half and quarter rate respectively.

The waveforms show how the AFI command phase corresponds to the memory command output. AFI command 0 corresponds to the first memory command slot, AFI command 1 corresponds to the second memory command slot, and so on.

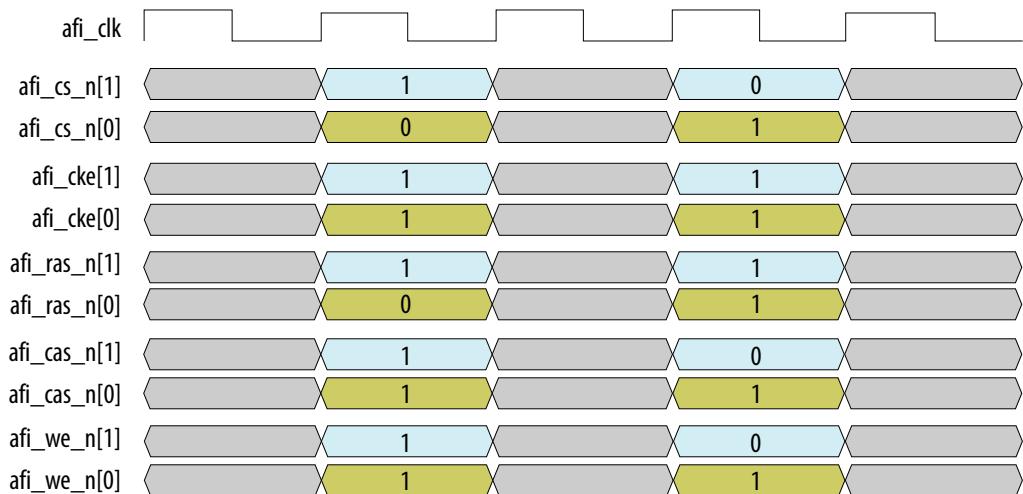
**Figure 117. AFI Address and Command Full-Rate**
**Memory Interface**

**AFI Interface**


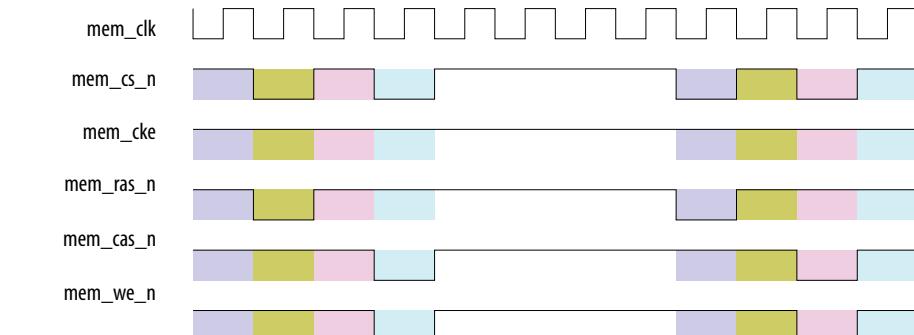
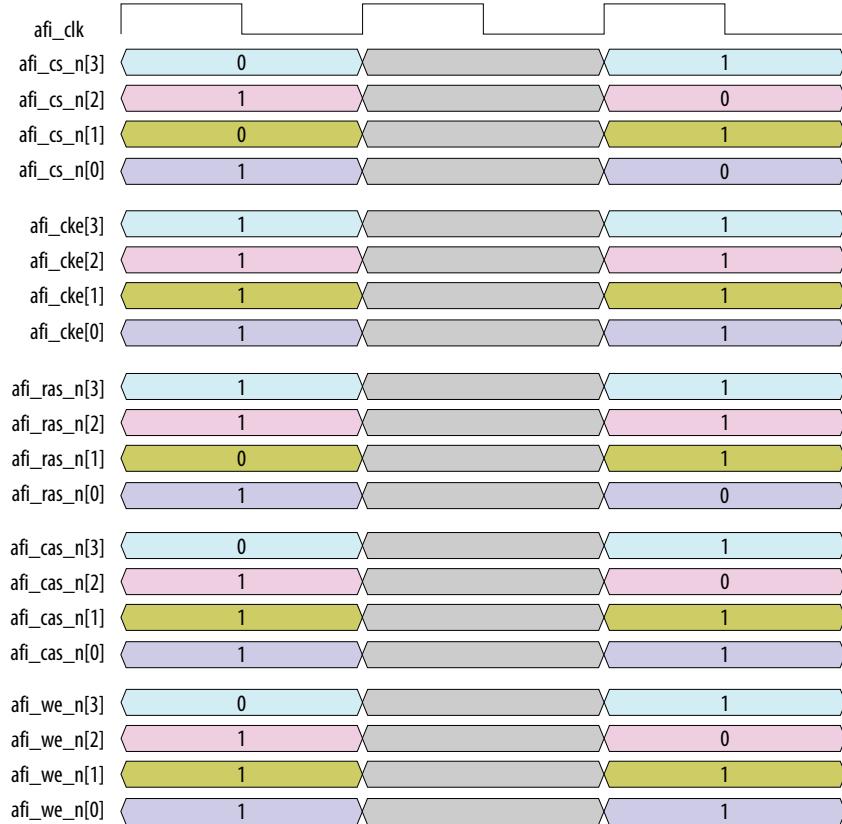


**Figure 118. AFI Address and Command Half-Rate**  
**Memory Interface**



**AFI Interface**



**Figure 119. AFI Address and Command Quarter-Rate**
**Memory Interface**

**AFI Interface**

**3.18.4.2. AFI Write Sequence Timing Diagrams**

The following timing diagrams illustrate the relationships between the write command and corresponding write data and write enable signals, in full, half, and quarter rate.

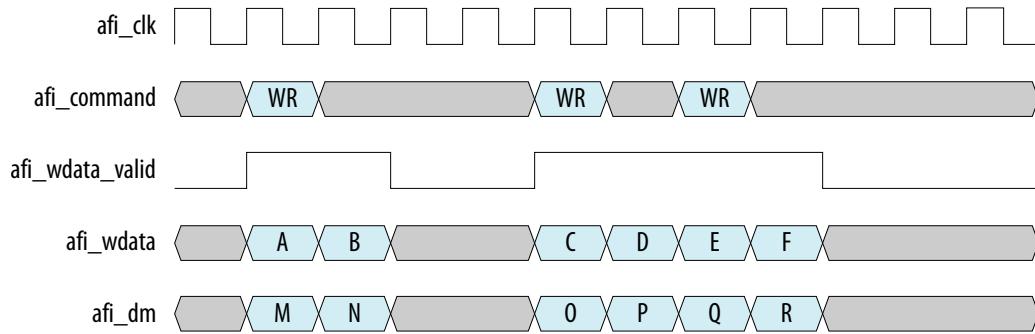
For half rate and quarter rate, when the write command is sent on the first memory clock in a PHY clock (for example,  $\text{afi\_cs\_n}[0] = 0$ ), that access is called *aligned* access; otherwise it is called *unaligned* access. You may use either aligned or unaligned access, or you may use both, but you must ensure that the distance



between the write command and the corresponding write data are constant on the AFI interface. For example, if a command is sent on the second memory clock in a PHY clock, the write data must also start at the second memory clock in a PHY clock.

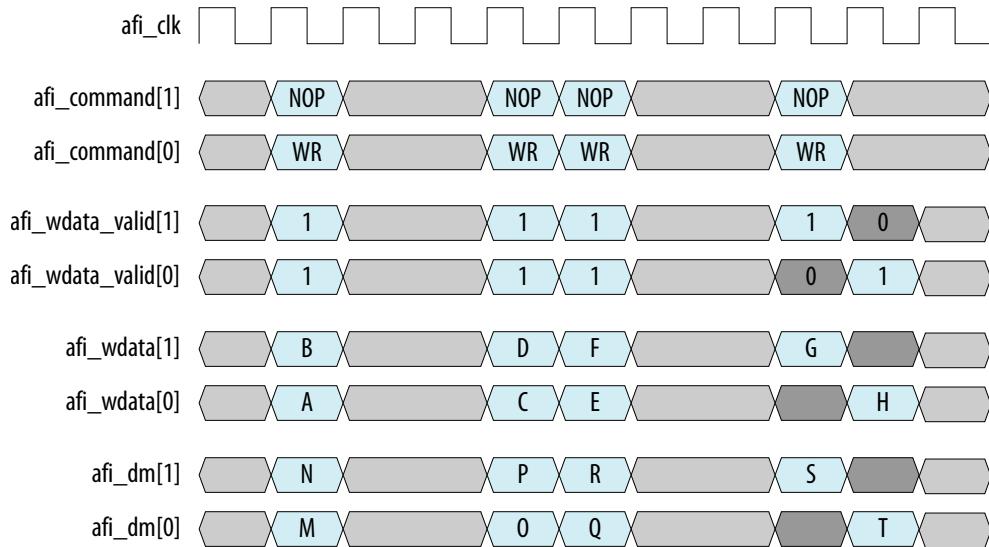
### Write sequences with wlat=0

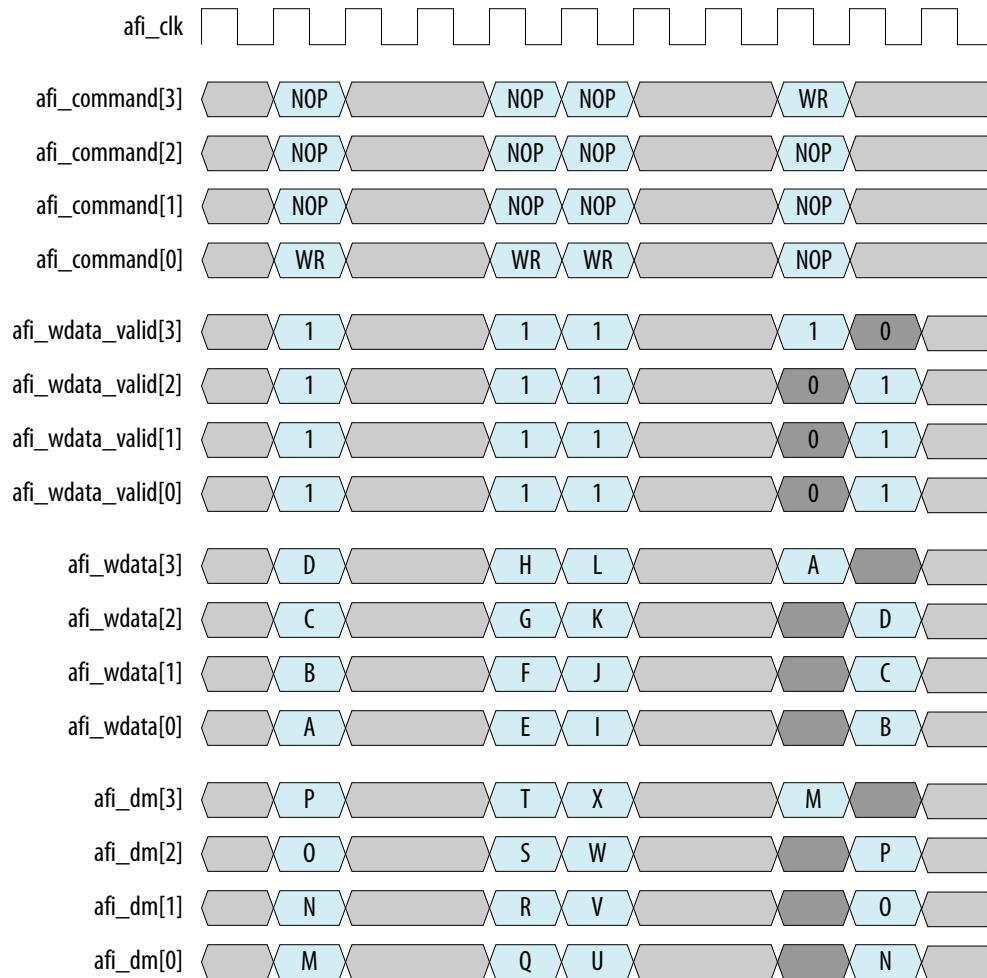
**Figure 120. AFI Write Data Full-Rate, wlat=0**



The following diagrams illustrate both aligned and unaligned access. The first three write commands are aligned accesses where they were issued on LSB of afi\_command. The fourth write command is unaligned access where it was issued on a different command slot. AFI signals must be shifted accordingly, based on the command slot.

**Figure 121. AFI Write Data Half-Rate, wlat=0**



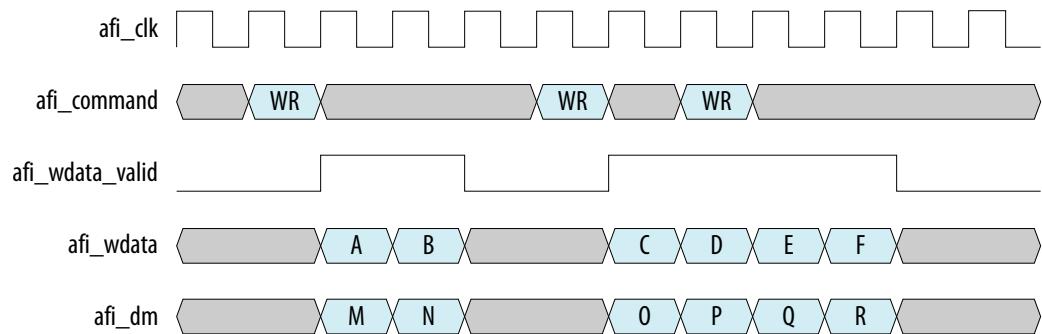
**Figure 122. AFI Write Data Quarter-Rate, wlat=0**


#### Write sequences with `wlat=non-zero`

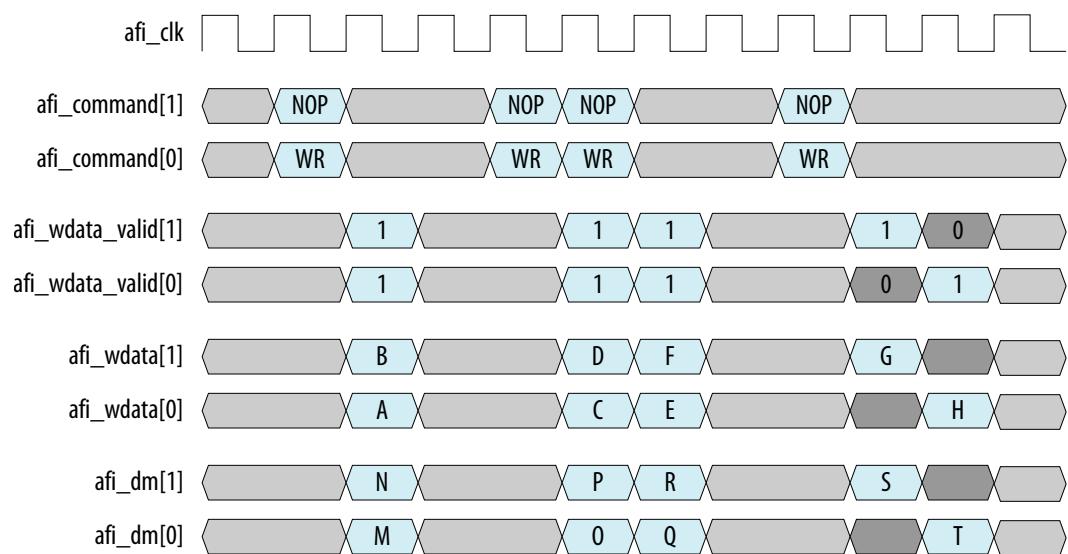
The `afi_wlat` is a signal from the PHY. The controller must delay `afi_dqs_burst`, `afi_wdata_valid`, `afi_wdata` and `afi_dm` signals by a number of PHY clock cycles equal to `afi_wlat`, which is a static value determined by calibration before the PHY asserts `cal_success` to the controller. The following figures illustrate the cases when `wlat=1`. Note that `wlat` is in the number of PHY clocks and therefore `wlat=1` equals 1, 2, and 4 memory clocks delay, respectively, on full, half and quarter rate.

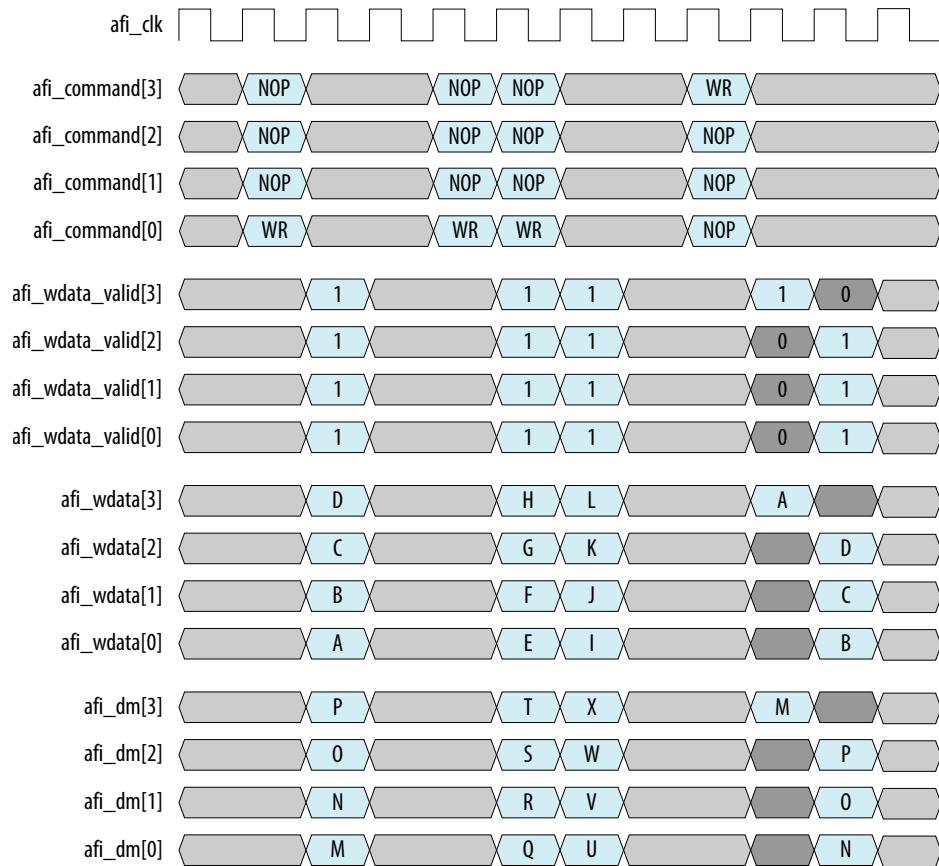


**Figure 123. AFI Write Data Full-Rate, wlat=1**



**Figure 124. AFI Write Data Half-Rate, wlat=1**



**Figure 125. AFI Write Data Quarter-Rate, wlat=1**


### DQS burst

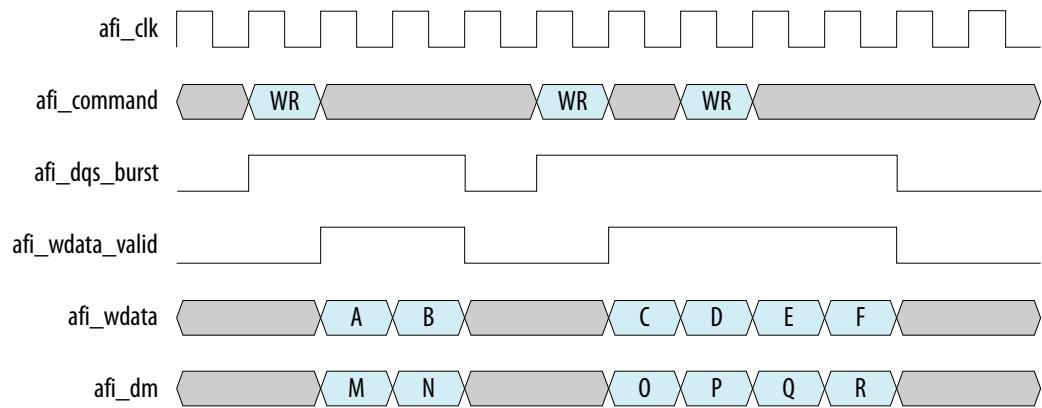
The **afi\_dqs\_burst** signal must be asserted one or two complete memory clock cycles earlier to generate DQS preamble. DQS preamble is equal to one-half and one-quarter AFI clock cycles in half and quarter rate, respectively.

A DQS preamble of two is required in DDR4, when the write preamble is set to two clock cycles.

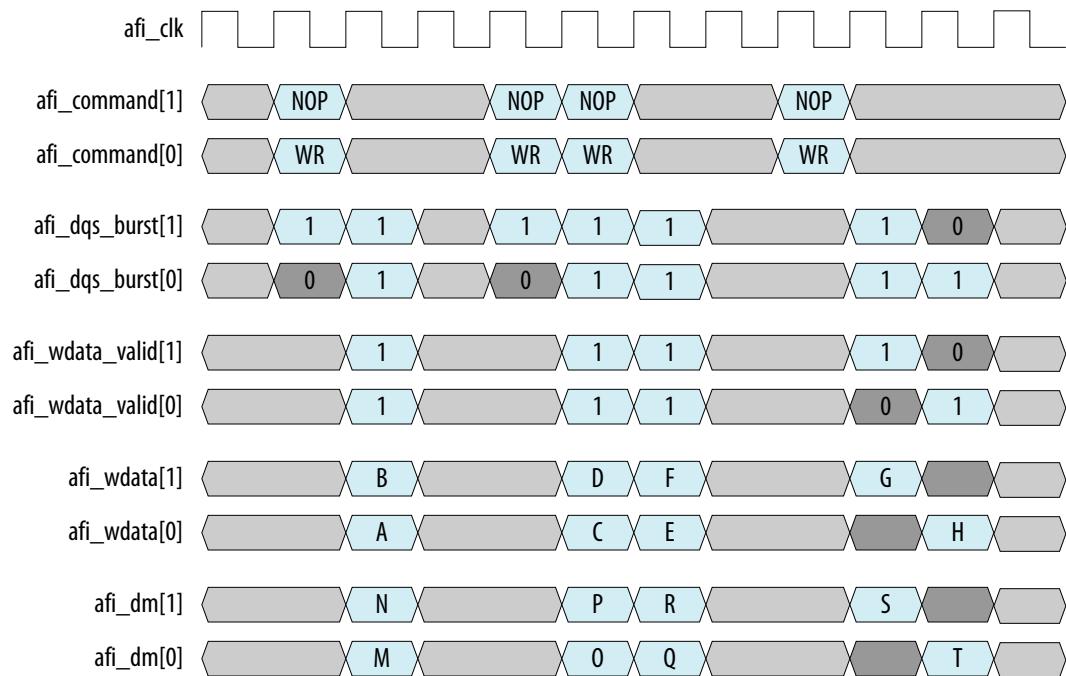
The following diagrams illustrate how **afi\_dqs\_burst** must be asserted in full, half, and quarter-rate configurations.

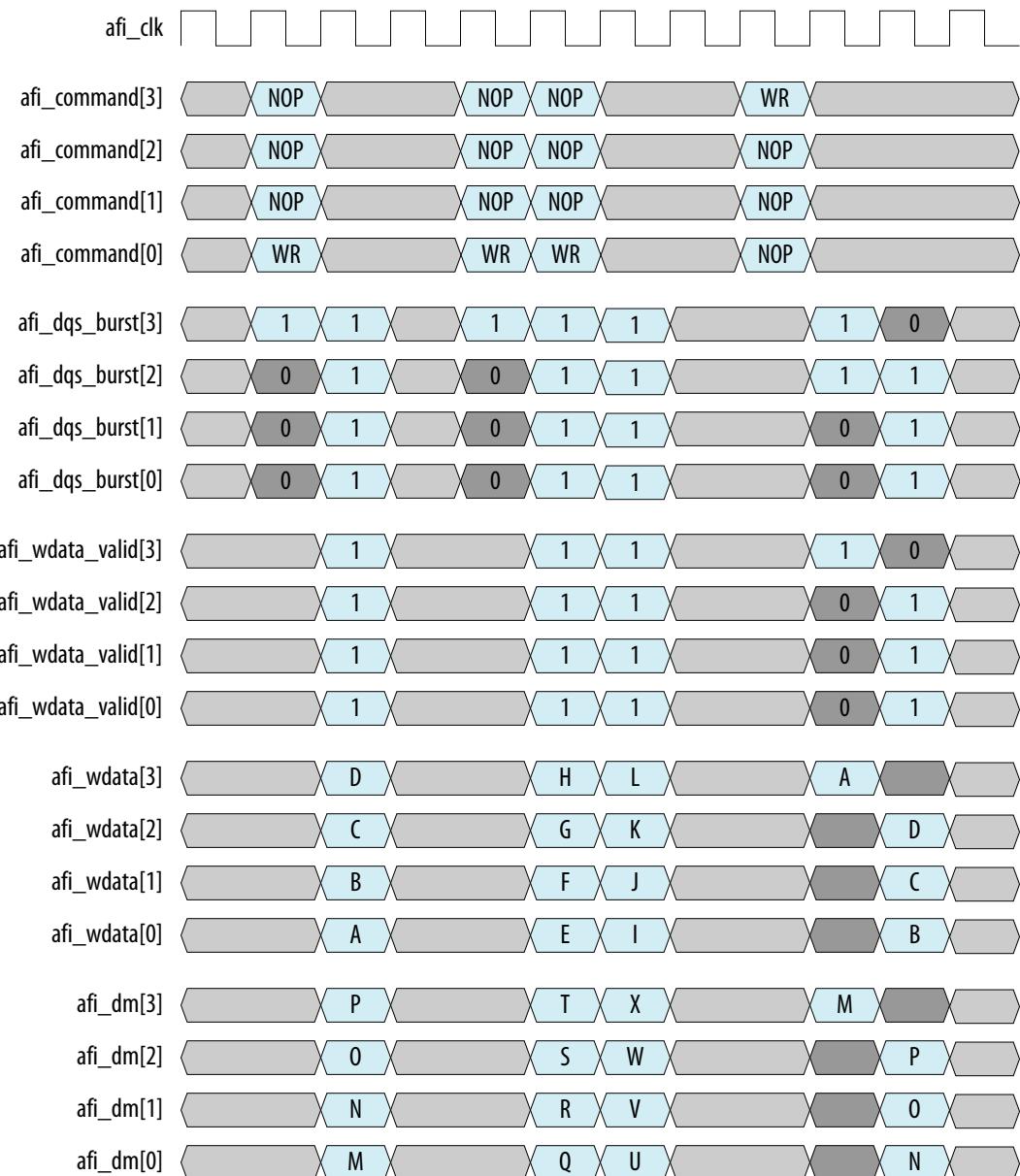


**Figure 126. AFI DQS Burst Full-Rate, wlat=1**



**Figure 127. AFI DQS Burst Half-Rate, wlat=1**



**Figure 128. AFI DQS Burst Quarter-Rate, wlat=1**


#### **Write data sequence with DBI (DDR4 and QDRIV only)**

The DDR4 write DBI feature is supported in the PHY, and when it is enabled, the PHY sends and receives the DBI signal without any controller involvement. The sequence is identical to non-DBI scenarios on the AFI interface.

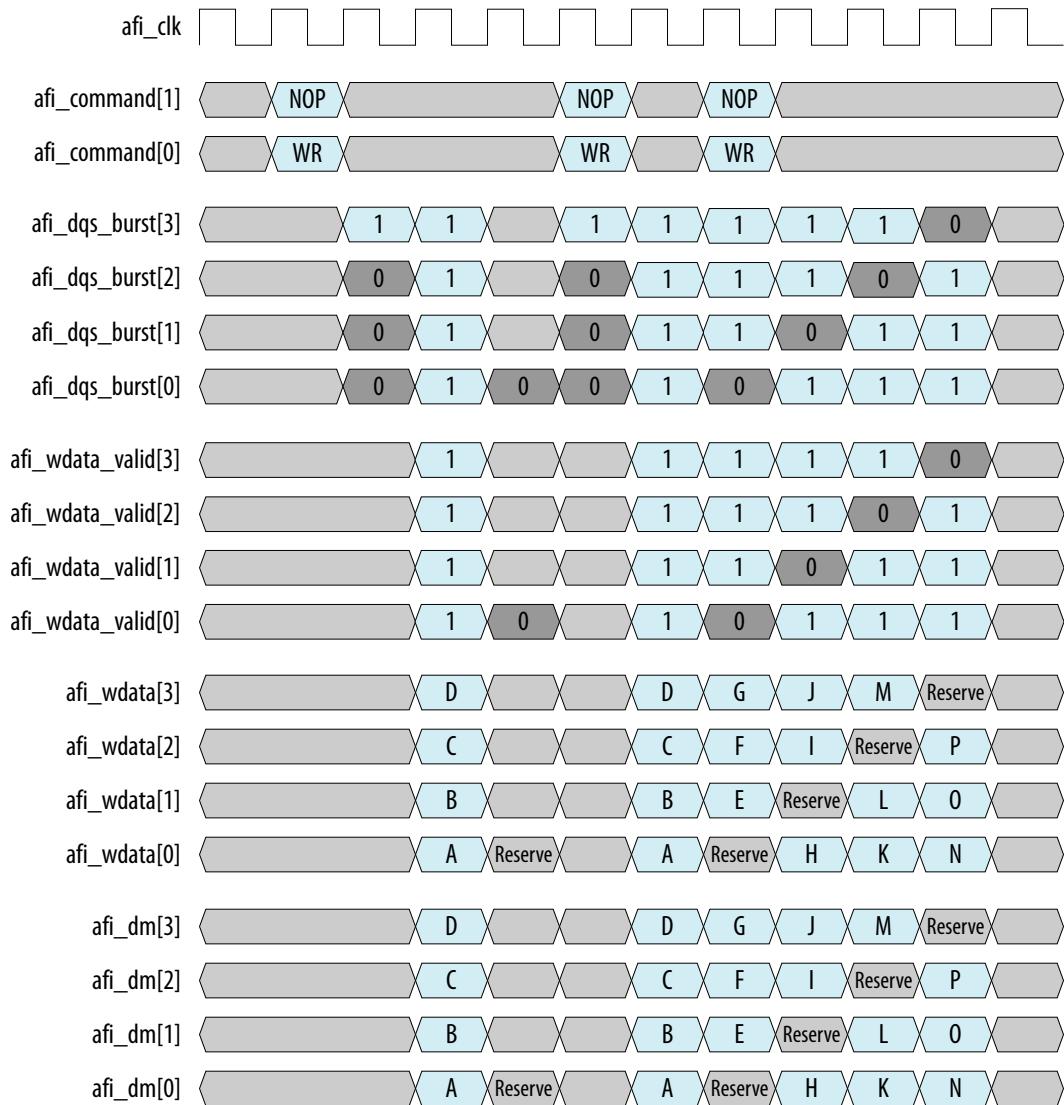
## Write data sequence with CRC (DDR4 only)

When the CRC feature of the PHY is enabled and used, the controller ensures at least one memory clock cycle between write commands, during which the PHY inserts the CRC data. Sending back to back write command would cause functional failure. The following figures show the legal sequences in CRC mode.

Entries marked as *0* and *RESERVE* must be observed by the controller; no information is allowed on those entries.

**Figure 129. AFI Write Data with CRC Half-Rate, wlat=2**



**Figure 130. AFI Write Data with CRC Quarter-Rate, wlat=2**


### 3.18.4.3. AFI Read Sequence Timing Diagrams

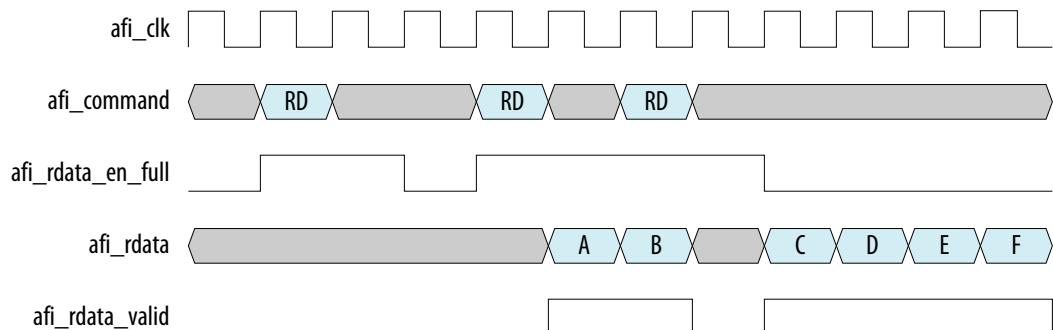
The following waveforms illustrate the AFI write data waveform in full, half, and quarter-rate, respectively.

The `afi_rdata_en_full` signal must be asserted for the entire read burst operation. The `afi_rdata_en` signal need only be asserted for the intended read data.

Aligned and unaligned access for read commands is similar to write commands; however, the `afi_rdata_en_full` signal must be sent on the same memory clock in a PHY clock as the read command. That is, if a read command is sent on the second memory clock in a PHY clock, `afi_rdata_en_full` must also be asserted, starting from the second memory clock in a PHY clock.

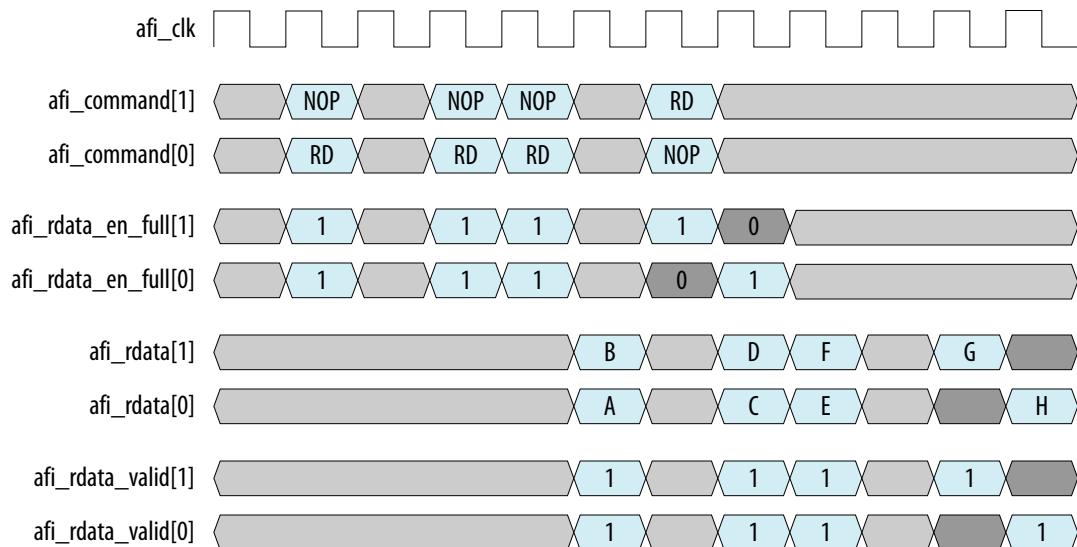


**Figure 131. AFI Read Data Full-Rate**

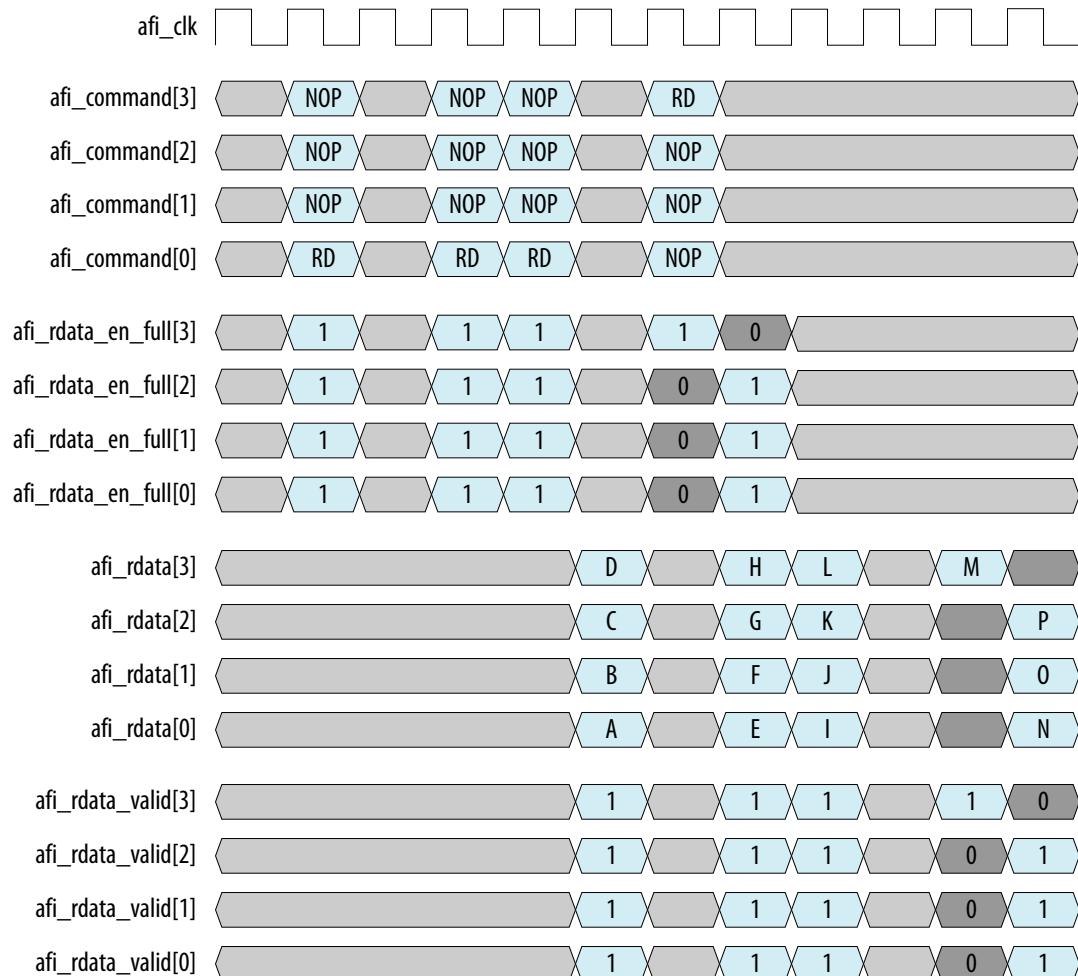


The following figure illustrates that the second and third reads require only the first and second half of data, respectively. The first three read commands are aligned accesses where they are issued on the LSB of afi\_command. The fourth read command is unaligned access, where it is issued on a different command slot. AFI signals must be shifted accordingly, based on command slot.

**Figure 132. AFI Read Data Half-Rate**



In the following figure, the first three read commands are aligned accesses where they are issued on the LSB of afi\_command. The fourth read command is unaligned access, where it is issued on a different command slot. AFI signals must be shifted accordingly, based on command slot.

**Figure 133. AFI Read Data Quarter-Rate**


#### 3.18.4.4. AFI Calibration Status Timing Diagram

The controller interacts with the PHY during calibration at power-up and at recalibration.

At power-up, the PHY holds `afi_cal_success` and `afi_cal_fail` 0 until calibration is done, when it asserts `afi_cal_success`, indicating to controller that the PHY is ready to use and `afi_wlat` and `afi_rlat` signals have valid values.

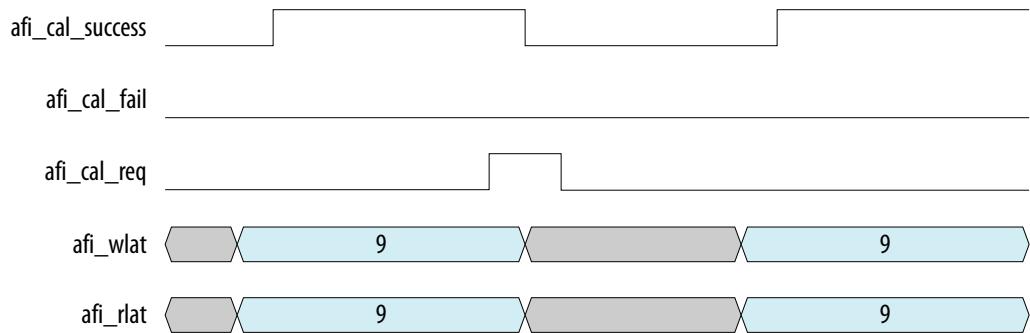
At recalibration, the controller asserts `afi_cal_req`, which triggers the same sequence as at power-up, and forces recalibration of the PHY.



**Figure 134. Calibration**



### AFI Interface



## 3.19. Resource Utilization

The following tables provide resource utilization information for external memory interfaces on Arria 10 devices.

### 3.19.1. QDR-IV Resource Utilization in Arria 10 Devices

The following table shows typical resource usage of QDR-IV interfaces with soft controller for Arria 10 devices.

**Table 56. QDR-IV Resource Utilization in Arria 10 Devices**

| Memory Width (Bits) | Combinational ALUTs | Dedicated Logic Registers | Block Memory Bits | M20Ks | Soft Controller |
|---------------------|---------------------|---------------------------|-------------------|-------|-----------------|
| 18                  | 2123                | 4592                      | 18432             | 8     | 1               |
| 36                  | 2127                | 6023                      | 36864             | 16    | 1               |
| 72                  | 2114                | 8826                      | 73728             | 32    | 1               |

## 3.20. Arria 10 EMIF Latency

The following latency data applies to all memory protocols supported by the Arria 10 EMIF IP.

**Table 57. Latency in Full-Rate Memory Clock Cycles**

| Rate <sup>1</sup> | Controller Address & Command | PHY Address & Command | Memory Read Latency <sup>2</sup> | PHY Read Data Return | Controller Read Data Return | Round Trip | Round Trip Without Memory |
|-------------------|------------------------------|-----------------------|----------------------------------|----------------------|-----------------------------|------------|---------------------------|
| Half:Write        | 12                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Half:Read         | 8                            | 2                     | 3-23                             | 6                    | 8                           | 27-47      | 24                        |

*continued...*



| Rate <sup>1</sup>   | Controller Address & Command | PHY Address & Command | Memory Read Latency <sup>2</sup> | PHY Read Data Return | Controller Read Data Return | Round Trip | Round Trip Without Memory |
|---------------------|------------------------------|-----------------------|----------------------------------|----------------------|-----------------------------|------------|---------------------------|
| Quarter:Write       | 14                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Quarter:Read        | 10                           | 2                     | 3-23                             | 6                    | 14                          | 35-55      | 32                        |
| Half:Write (ECC)    | 14                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Half:Read (ECC)     | 12                           | 2                     | 3-23                             | 6                    | 8                           | 31-51      | 28                        |
| Quarter:Write (ECC) | 14                           | 2                     | 3-23                             | —                    | —                           | —          | —                         |
| Quarter:Read (ECC)  | 12                           | 2                     | 3-23                             | 6                    | 14                          | 37-57      | 34                        |

1. User interface rate; the controller always operates in half rate.  
2. Minimum and maximum read latency range for DDR3, DDR4, and LPDDR3.

### 3.21. Arria 10 EMIF Calibration Times

The time needed for calibration varies, depending on many factors including the interface width, the number of ranks, frequency, board layout, and difficulty of calibration.

The following table lists approximate typical calibration times for various protocols and configurations.

**Table 58. Arria 10 EMIF IP Approximate Calibration Times**

| Protocol                        | Rank and Frequency | Typical Calibration Time |
|---------------------------------|--------------------|--------------------------|
| DDR3, x64 UDIMM, DQS x8, DM on  | 1 rank, 933 MHz    | 102 ms                   |
|                                 | 1 rank, 800 MHz    | 106 ms                   |
|                                 | 2 rank, 933 MHz    | 198 ms                   |
|                                 | 2 rank, 800 MHz    | 206 ms                   |
| DDR4, x64 UDIMM, DQS x8, DBI on | 1 rank, 1067 MHz   | 314 ms                   |
|                                 | 1 rank, 800 MHz    | 353 ms                   |
|                                 | 2 rank 1067 MHz    | 625 ms                   |
|                                 | 2 rank 800 MHz     | 727 ms                   |
| RLDRAM 3, x36                   | 1200 MHz           | 2808 ms                  |
|                                 | 1067 MHz           | 2825 ms                  |
|                                 | 1200 MHz, with DM  | 2818 ms                  |
|                                 | 1067 MHz, with DM  | 2833 ms                  |
| QDR II, x36, BWS on             | 333 MHz            | 616 ms                   |

*continued...*



| Protocol            | Rank and Frequency | Typical Calibration Time |
|---------------------|--------------------|--------------------------|
|                     | 633 MHz            | 833 ms                   |
| QDR-IV, x36, BWS on | 1067 MHz           | 1563 ms                  |
|                     | 1067 MHz, with DBI | 1556 ms                  |

## 3.22. Integrating a Custom Controller with the Hard PHY

If you want to use your own custom memory controller, you must integrate the controller with the hard PHY to achieve a complete memory solution.

Observe the following general guidelines:

- When you configure your external memory interface IP, ensure that you select **Configuration > Hard PHY Only** on the **General** tab in the parameter editor.
- Consult the AFI 4.0 Specification, for detailed information on the AFI interface to the PHY.

## 3.23. Memory Mapped Register (MMR) Tables

The address buses to read and write from the MMR registers are 10 bits wide, while the read and write data buses are configured to be 32 bits. The Bits Register Link column in the table below provides the mapping on the width of the data read within the 32-bit bus. The reads and writes are always performed using the 32-bit-wide bus.

**Register Summary**

| Register    | Address 32-bit Bus | Bits Register link |
|-------------|--------------------|--------------------|
| sbcfg8      | 7                  | 16                 |
| sbcfg9      | 8                  | 16                 |
| reserve2    | 9                  | 16                 |
| ctrlcfg0    | 10                 | 32                 |
| ctrlcfg1    | 11                 | 32                 |
| ctrlcfg2    | 12                 | 32                 |
| ctrlcfg3    | 13                 | 32                 |
| ctrlcfg4    | 14                 | 32                 |
| ctrlcfg5    | 15                 | 16                 |
| ctrlcfg6    | 16                 | 16                 |
| ctrlcfg7    | 17                 | 16                 |
| ctrlcfg8    | 18                 | 8                  |
| ctrlcfg9    | 19                 | 8                  |
| dramtiming0 | 20                 | 24                 |
| dramodt0    | 21                 | 32                 |
| dramodt1    | 22                 | 24                 |

*continued...*



| Register    | Address 32-bit Bus | Bits Register link |
|-------------|--------------------|--------------------|
| sbcfg0      | 23                 | 32                 |
| sbcfg1      | 24                 | 32                 |
| sbcfg2      | 25                 | 8                  |
| sbcfg3      | 26                 | 24                 |
| sbcfg4      | 27                 | 24                 |
| sbcfg5      | 28                 | 8                  |
| sbcfg6      | 29                 | 32                 |
| sbcfg7      | 30                 | 8                  |
| caltiming0  | 31                 | 32                 |
| caltiming1  | 32                 | 32                 |
| caltiming2  | 33                 | 32                 |
| caltiming3  | 34                 | 32                 |
| caltiming4  | 35                 | 32                 |
| caltiming5  | 36                 | 24                 |
| caltiming6  | 37                 | 32                 |
| caltiming7  | 38                 | 32                 |
| caltiming8  | 39                 | 32                 |
| caltiming9  | 40                 | 8                  |
| caltiming10 | 41                 | 8                  |
| dramaddrw   | 42                 | 24                 |
| sideband0   | 43                 | 8                  |
| sideband1   | 44                 | 8                  |
| sideband2   | 45                 | 8                  |
| sideband3   | 46                 | 8                  |
| sideband4   | 47                 | 8                  |
| sideband5   | 48                 | 8                  |
| sideband6   | 49                 | 8                  |
| sideband7   | 50                 | 8                  |
| sideband8   | 51                 | 8                  |
| sideband9   | 52                 | 8                  |
| sideband10  | 53                 | 8                  |
| sideband11  | 54                 | 8                  |
| sideband12  | 55                 | 8                  |
| sideband13  | 56                 | 32                 |
| sideband14  | 57                 | 16                 |
| sideband15  | 58                 | 8                  |

*continued...*



| Register      | Address 32-bit Bus | Bits Register link |
|---------------|--------------------|--------------------|
| dramsts       | 59                 | 8                  |
| dbgdone       | 60                 | 8                  |
| dbgsignals    | 61                 | 32                 |
| dbgreset      | 62                 | 8                  |
| dbgmatch      | 63                 | 32                 |
| counter0mask  | 64                 | 32                 |
| counter1mask  | 65                 | 32                 |
| counter0match | 66                 | 32                 |
| counter1match | 67                 | 32                 |
| niosreserve0  | 68                 | 16                 |
| niosreserve1  | 69                 | 16                 |
| niosreserve2  | 70                 | 16                 |
| ecc1          | 128                | 10                 |
| ecc2          | 129                | 22                 |
| ecc3          | 130                | 9                  |
| ecc4          | 144                | 16                 |

Note: Addresses are in decimal format.

### 3.23.1. ctrlcfg0: Controller Configuration

**address=10(32 bit)**

| Field                 | Bit High | Bit Low | Description  | Access     |
|-----------------------|----------|---------|--|------------|
| cfg_mem_type          | 3        | 0       | Selects memory type. Program this field with one of the following binary values, "0000" for DDR3 SDRAM, "0001" for DDR4 SDRAM, "0010" for LPDDR3 SDRAM and "0011" for RLDRAM3.   | Read/Write |
| cfg_dimm_type         | 6        | 4       | Selects dimm type. Program this field with one of the following binary values: 3' b000: for DIMM_TYPE_COMPONENT, 3' b001: for DIMM_TYPE_UDIMM, 3' b010: for DIMM_TYPE_RDIMM, 3' b011: for DIMM_TYPE_LRDIMM, 3' b100: for DIMM_TYPE_SODIMM, and 3' b101: for DIMM_TYPE_3DS. | Read/Write |
| cfg_ac_pos            | 8        | 7       | Specify C/A (command/address) pin position.  | Read/Write |
| cfg_ctrl_burst_length | 13       | 9       | Configures burst length for control path. Legal values are valid for JEDEC allowed DRAM values for the DRAM selected in cfg_type. For DDR3, DDR4   | Read/Write |

**continued...**



| Field                 | Bit High | Bit Low | Description   | Access     |
|-----------------------|----------|---------|---|------------|
|                       |          |         | and LPDDR3, this should be programmed with 8 (binary "01000"), for RLDRAM III it can be programmed with 2 or 4 or 8.  |            |
| cfg_dbc0_burst_length | 18       | 14      | Configures burst length for DBC0. Legal values are valid for JEDEC allowed DRAM values for the DRAM selected in cfg_type. For DDR3, DDR4 and LPDDR3, this should be programmed with 8 (binary "01000"), for RLDRAM III it can be programmed with 2 or 4 or 8. | Read/Write |
| cfg_dbc1_burst_length | 23       | 19      | Configures burst length for DBC1. Legal values are valid for JEDEC allowed DRAM values for the DRAM selected in cfg_type. For DDR3, DDR4 and LPDDR3, this should be programmed with 8 (binary "01000"), for RLDRAM III it can be programmed with 2 or 4 or 8. | Read/Write |
| cfg_dbc2_burst_length | 28       | 24      | Configures burst length for DBC2. Legal values are valid for JEDEC allowed DRAM values for the DRAM selected in cfg_type. For DDR3, DDR4 and LPDDR3, this should be programmed with 8 (binary "01000"), for RLDRAM III it can be programmed with 2 or 4 or 8. | Read/Write |

### 3.23.2. ctrlcfg1: Controller Configuration

address=11(32 bit)

| Field                 | Bit High | Bit Low | Description   | Access     |
|-----------------------|----------|---------|---|------------|
| cfg_dbc3_burst_length | 4        | 0       | Configures burst length for DBC3. Legal values are valid for JEDEC allowed DRAM values for the DRAM selected in cfg_type. For DDR3, DDR4 and LPDDR3, this should be programmed with 8 (binary "01000"), for RLDRAM III it can be programmed with 2 or 4 or 8.   | Read/Write |
| cfg_addr_order        | 6        | 5       | Selects the order for address interleaving. Programming this field with different values gives different mappings between the AXI or Avalon-MM address and the SDRAM address. Program this field with the following binary values to select the ordering. "00" - chip, row, bank(BG, BA), column; "01" - chip, bank(BG, BA), row, column; "10"-row, chip, bank(BG, BA), column. | Read/Write |
| cfg_ctrl_enable_ec    | 7        | 7       | Enable the generation and checking of ECC.  | Read/Write |
| cfg_dbc0_enable_ecc   | 8        | 8       | Enable the generation and checking of ECC.  | Read/Write |

*continued...*



| Field                  | Bit High | Bit Low | Description   | Access     |
|------------------------|----------|---------|---|------------|
| cfg_dbc1_enable_ecc    | 9        | 9       | Enable the generation and checking of ECC.  | Read/Write |
| cfg_dbc2_enable_ecc    | 10       | 10      | Enable the generation and checking of ECC.  | Read/Write |
| cfg_dbc3_enable_ecc    | 11       | 11      | Enable the generation and checking of ECC.  | Read/Write |
| cfg_reorder_data       | 12       | 12      | This bit controls whether the controller can re-order operations to optimize SDRAM bandwidth. It should generally be set to a one.                                  | Read/Write |
| cfg_ctrl_reorder_rdata | 13       | 13      | This bit controls whether the controller need to re-order the read return data.   | Read/Write |
| cfg_dbc0_reorder_rdata | 14       | 14      | This bit controls whether the controller need to re-order the read return data.   | Read/Write |
| cfg_dbc1_reorder_rdata | 15       | 15      | This bit controls whether the controller need to re-order the read return data.   | Read/Write |
| cfg_dbc2_reorder_rdata | 16       | 16      | This bit controls whether the controller need to re-order the read return data.   | Read/Write |
| cfg_dbc3_reorder_rdata | 17       | 17      | This bit controls whether the controller need to re-order the read return data.   | Read/Write |
| cfg_reorder_read       | 18       | 18      | This bit controls whether the controller can re-order read command to 1.  | Read/Write |
| cfg_starve_limit       | 24       | 19      | Specifies the number of DRAM burst transactions an individual transaction will allow to reorder ahead of it before its priority is raised in the memory controller. | Read/Write |
| cfg_dqstrk_en          | 25       | 25      | Enables DQS tracking in the PHY.  | Read/Write |
| cfg_ctrl_enable_dm     | 26       | 26      | Set to a one to enable DRAM operation if DM pins are connected.   | Read/Write |
| cfg_dbc0_enable_dm     | 27       | 27      | Set to a one to enable DRAM operation if DM pins are connected.   | Read/Write |
| cfg_dbc1_enable_dm     | 28       | 28      | Set to a one to enable DRAM operation if DM pins are connected.   | Read/Write |
| cfg_dbc2_enable_dm     | 29       | 29      | Set to a one to enable DRAM operation if DM pins are connected.   | Read/Write |
| cfg_dbc3_enable_dm     | 30       | 30      | Set to a one to enable DRAM operation if DM pins are connected.   | Read/Write |

### 3.23.3. ctrlcfg2: Controller Configuration

**address=12(32 bit)**

| Field                | Bit High | Bit Low | Description  | Access     |
|----------------------|----------|---------|--|------------|
| cfg_ctrl_output_regd | 0        | 0       | Set to one to register the HMC command output. Set to 0 to disable it.   | Read/Write |
| cfg_dbc0_output_regd | 1        | 1       | Set to one to register the HMC command output. Set to 0 to disable it.   | Read/Write |
| cfg_dbc1_output_regd | 2        | 2       | Set to one to register the HMC command output. Set to 0 to disable it.   | Read/Write |
| cfg_dbc2_output_regd | 3        | 3       | Set to one to register the HMC command output. Set to 0 to disable it.   | Read/Write |
| cfg_dbc3_output_regd | 4        | 4       | Set to one to register the HMC command output. Set to 0 to disable it.   | Read/Write |
| cfg_ctrl2dbc_switch0 | 6        | 5       | Select of the MUX ctrl2dbc_switch0. 2  | Read/Write |
| cfg_ctrl2dbc_switch1 | 8        | 7       | Select of the MUX ctrl2dbc_switch1. 2  | Read/Write |
| cfg_dbc0_ctrl_sel    | 9        | 9       | DBC0 - control path select. 1  | Read/Write |
| cfg_dbc1_ctrl_sel    | 10       | 10      | DBC1 - control path select. 1  | Read/Write |
| cfg_dbc2_ctrl_sel    | 11       | 11      | DBC2 - control path select. 1  | Read/Write |
| cfg_dbc3_ctrl_sel    | 12       | 12      | DBC3 - control path select. 1  | Read/Write |
| cfg_dbc2ctrl_sel     | 14       | 13      | Specifies which DBC is driven by the local control path. 2   | Read/Write |
| cfg_dbc0_pipe_lat    | 17       | 15      | Specifies in number of controller clock cycles the latency of pipelining the signals from control path to DBC0 | Read/Write |
| cfg_dbc1_pipe_lat    | 20       | 18      | Specifies in number of controller clock cycles the latency of pipelining the signals from control path to DBC1 | Read/Write |
| cfg_dbc2_pipe_lat    | 23       | 21      | Specifies in number of controller clock cycles the latency of pipelining the signals from control path to DBC2 | Read/Write |
| cfg_dbc3_pipe_lat    | 26       | 24      | Specifies in number of controller clock cycles the latency of pipelining the signals from control path to DBC3 | Read/Write |

**3.23.4. ctrlcfg3: Controller Configuration****address=13(32 bit)**

| Field                      | Bit High | Bit Low | Description | Access     |
|----------------------------|----------|---------|-------------|------------|
| cfg_ctrl_cmd_rate          | 2        | 0       | 3           | Read/Write |
| cfg_dbc0_cmd_rate          | 5        | 3       | 3           | Read/Write |
| cfg_dbc1_cmd_rate          | 8        | 6       | 3           | Read/Write |
| cfg_dbc2_cmd_rate          | 11       | 9       | 3           | Read/Write |
| <b><i>continued...</i></b> |          |         |             |            |



| Field                   | Bit High | Bit Low | Description  | Access     |
|-------------------------|----------|---------|--|------------|
| cfg_dbc3_cmd_rate       | 14       | 12      | 3  | Read/Write |
| cfg_ctrl_in_protocol    | 15       | 15      | 1  | Read/Write |
| cfg_dbc0_in_protocol    | 16       | 16      | 1  | Read/Write |
| cfg_dbc1_in_protocol    | 17       | 17      | 1  | Read/Write |
| cfg_dbc2_in_protocol    | 18       | 18      | 1  | Read/Write |
| cfg_dbc3_in_protocol    | 19       | 19      | 1  | Read/Write |
| cfg_ctrl_dualport_en    | 20       | 20      | Enable the second command port for RLDRAM3 only (BL=2 or 4)  | Read/Write |
| cfg_dbc0_dualport_en    | 21       | 21      | Enable the second data port for RLDRAM3 only (BL=2 or 4)   | Read/Write |
| cfg_dbc1_dualport_en    | 22       | 22      | Enable the second data port for RLDRAM3 only (BL=2 or 4)   | Read/Write |
| cfg_dbc2_dualport_en    | 23       | 23      | Enable the second data port for RLDRAM3 only (BL=2 or 4)   | Read/Write |
| cfg_dbc3_dualport_en    | 24       | 24      | Enable the second data port for RLDRAM3 only (BL=2 or 4)   | Read/Write |
| cfg_arbiter_type        | 25       | 25      | Indicates controller arbiter operating mode. Set this to: - 1  | Read/Write |
| cfg_open_page_en        | 26       | 26      | Set to 1 to enable the open page policy when command reordering is disabled (cfg_cmd_reorder = 0). This bit does not matter when cfg_cmd_reorder is 1. | Read/Write |
| cfg_rld3_multibank_mode | 30       | 28      | Multibank setting, specific for RLDRAM3. Set this to: - 3  | Read/Write |

Note: DDR4 gear down mode is not supported.

### 3.23.5. ctrlcfg4: Controller Configuration

**address=14(32 bit)**

| Field                   | Bit High | Bit Low | Description                                       | Access     |
|-------------------------|----------|---------|---|------------|
| cfg_tile_id             | 4        | 0       | Tile ID.  | Read/Write |
| cfg_pingpong_mode       | 6        | 5       | Ping Pong mode: 2                                 | Read/Write |
| cfg_ctrl_slot_rotate_en | 9        | 7       | Cmd slot rotate enable: bit[0] controls write, 1  | Read/Write |
| cfg_dbc0_slot_rotate_en | 12       | 10      | DBC0 slot rotate enable: bit[0] controls write, 1 | Read/Write |
| cfg_dbc1_slot_rotate_en | 15       | 13      | DBC1 slot rotate enable: bit[0] controls write, 1 | Read/Write |
| cfg_dbc2_slot_rotate_en | 18       | 16      | DBC2 slot rotate enable: bit[0] controls write, 1 | Read/Write |
| cfg_dbc3_slot_rotate_en | 21       | 19      | DBC3 slot rotate enable: bit[0] controls write, 1 | Read/Write |

*continued...*



| Field                | Bit High | Bit Low | Description  | Access     |
|----------------------|----------|---------|--|------------|
| cfg_ctrl_slot_offset | 23       | 22      | Enables afi information to be offset by numbers of FR cycles. Affected afi signal is afi_rdata_en, afi_rdata_en_full, afi_wdata_valid, afi_dqs_burst, afi_mrnk_write and afi_mrnk_read. Set this to: - 2 | Read/Write |
| cfg_dbc0_slot_offset | 25       | 24      | Enables afi information to be offset by numbers of FR cycles. Affected afi signal is afi_rdata_en, afi_rdata_en_full, afi_wdata_valid, afi_dqs_burst, afi_mrnk_write and afi_mrnk_read. Set this to: - 2 | Read/Write |
| cfg_dbc1_slot_offset | 27       | 26      | Enables afi information to be offset by numbers of FR cycles. Affected afi signal is afi_rdata_en, afi_rdata_en_full, afi_wdata_valid, afi_dqs_burst, afi_mrnk_write and afi_mrnk_read. Set this to: - 2 | Read/Write |
| cfg_dbc2_slot_offset | 29       | 28      | Enables afi information to be offset by numbers of FR cycles. Affected afi signal is afi_rdata_en, afi_rdata_en_full, afi_wdata_valid, afi_dqs_burst, afi_mrnk_write and afi_mrnk_read. Set this to: - 2 | Read/Write |
| cfg_dbc3_slot_offset | 31       | 30      | Enables afi information to be offset by numbers of FR cycles. Affected afi signal is afi_rdata_en, afi_rdata_en_full, afi_wdata_valid, afi_dqs_burst, afi_mrnk_write and afi_mrnk_read. Set this to: - 2 | Read/Write |

### 3.23.6. ctrlcfg5: Controller Configuration

address=15(32 bit)

| Field            | Bit High | Bit Low | Description  | Access     |
|------------------|----------|---------|--|------------|
| cfg_col_cmd_slot | 3        | 0       | Specify the col cmd slot. One hot encoding.  | Read/Write |
| cfg_row_cmd_slot | 7        | 4       | Specify the row cmd slot. One hot encoding.  | Read/Write |
| cfg_ctrl_rc_en   | 8        | 8       | Set to 1 to enable the rate conversion. It converts QR input from core to HR inside HMC. | Read/Write |
| cfg_dbc0_rc_en   | 9        | 9       | Set to 1 to enable the rate conversion. It converts QR input from core to HR inside HMC. | Read/Write |
| cfg_dbc1_rc_en   | 10       | 10      | Set to 1 to enable the rate conversion. It converts QR input from core to HR inside HMC. | Read/Write |
| cfg_dbc2_rc_en   | 11       | 11      | Set to 1 to enable the rate conversion. It converts QR input from core to HR inside HMC. | Read/Write |
| cfg_dbc3_rc_en   | 12       | 12      | Set to 1 to enable the rate conversion. It converts QR input from core to HR inside HMC. | Read/Write |



### 3.23.7. ctrlcfg6: Controller Configuration

**address=16(32 bit)**

| Field       | Bit High | Bit Low | Description   | Access     |
|-------------|----------|---------|---|------------|
| cfg_cs_chip | 15       | 0       | Chip select mapping scheme. Mapping separated into 4 sections: [CS3][CS2] [CS1][CS0] Each section consists of 4 bits to indicate which CS_n signal should be active when command goes to current CS. Eg: if we set to 16. | Read/Write |

### 3.23.8. ctrlcfg7: Controller Configuration

**address=17(32 bit)**

| Field                 | Bit High | Bit Low | Description   | Access     |
|-----------------------|----------|---------|---|------------|
| cfg_clkgating_en      | 0        | 0       | Set to 1 to enable the clock gating. The clock is shut off for the whole HMC.         | Read/Write |
| cfg_rb_reserved_entry | 7        | 1       | Specify how many entries are reserved in read buffer before almost full is asserted.  | Read/Write |
| cfg_wb_reserved_entry | 14       | 8       | Specify how many entries are reserved in write buffer before almost full is asserted. | Read/Write |

### 3.23.9. ctrlcfg8: Controller Configuration

**address=18(32 bit)**

| Field            | Bit High | Bit Low | Description                                       | Access     |
|------------------|----------|---------|---|------------|
| cfg_3ds_en       | 0        | 0       | Setting to 1 to enable #DS support for DDR4.      | Read/Write |
| cfg_ck_inv       | 1        | 1       | Use to program CK polarity. 1                     | Read/Write |
| cfg_addr_mplx_en | 2        | 2       | Setting to 1 enables RLD3 address multiplex mode. | Read/Write |

### 3.23.10. ctrlcfg9: Controller Configuration

**address=19(32 bit)**

| Field             | Bit High | Bit Low | Description                                      | Access     |
|-------------------|----------|---------|--|------------|
| cfg_dfx_bypass_en | 0        | 0       | Used for dft and timing characterization only. 1 | Read/Write |



### 3.23.11. dramtiming0: Timing Parameters

**address=20(32 bit)**

| Field                            | Bit High | Bit Low | Description   | Access     |
|----------------------------------|----------|---------|---|------------|
| cfg_tcl                          | 6        | 0       | Memory read latency.  | Read/Write |
| cfg_power_saving_exit_cycles     | 12       | 7       | The minimum number of cycles to stay in a low power state. This applies to both power down and self-refresh and should be set to the greater of tPD and tCKESR.                         | Read/Write |
| cfg_mem_clk_disable_entry_cycles | 18       | 13      | Set to a the number of clocks after the execution of an self-refresh to stop the clock. This register is generally set based on PHY design latency and should generally not be changed. | Read/Write |

### 3.23.12. dramodt0: On-Die Termination Parameters

**address=21(32 bit)**

| Field              | Bit High | Bit Low | Description   | Access     |
|--------------------|----------|---------|---|------------|
| cfg_write_odt_chip | 15       | 0       | ODT scheme setting for write command. Setting separated into 4 sections: [CS3][CS2][CS1][CS0] Each section consists of 4 bits to indicate which chip should ODT be asserted when write occurs on current CS. Eg: if we set to 16. | Read/Write |
| cfg_read_odt_chip  | 31       | 16      | ODT scheme setting for read command. Setting separated into 4 sections: [CS3][CS2][CS1][CS0] Each section consists of 4 bits to indicate which chip should ODT be asserted when write occurs on current CS. Eg: if we set to 16.  | Read/Write |

### 3.23.13. dramodt1: On-Die Termination Parameters



#### address=22(32 bit)

| Field             | Bit High | Bit Low | Description   | Access     |
|-------------------|----------|---------|---|------------|
| cfg_wr_odt_on     | 5        | 0       | Indicates number of memory clock cycle gap between write command and ODT signal rising edge.    | Read/Write |
| cfg_rd_odt_on     | 11       | 6       | Indicates number of memory clock cycle gap between read command and ODT signal rising edge.     | Read/Write |
| cfg_wr_odt_period | 17       | 12      | Indicates number of memory clock cycle write ODT signal should stay asserted after rising edge. | Read/Write |
| cfg_rd_odt_period | 23       | 18      | Indicates number of memory clock cycle read ODT signal should stay asserted after rising edge.  | Read/Write |

#### 3.23.14. sbcfg0: Sideband Configuration

#### address=23(32 bit)

| Field                  | Bit High | Bit Low | Description   | Access     |
|------------------------|----------|---------|---|------------|
| cfg_rld3_refresh_seq_0 | 15       | 0       | Banks to Refresh for RLD3 in sequence 0. Must not be more than 4 banks. | Read/Write |
| cfg_rld3_refresh_seq_1 | 31       | 16      | Banks to Refresh for RLD3 in sequence 1. Must not be more than 4 banks. | Read/Write |

#### 3.23.15. sbcfg1: Sideband Configuration

#### address=24(32 bit)

| Field                  | Bit High | Bit Low | Description   | Access     |
|------------------------|----------|---------|---|------------|
| cfg_rld3_refresh_seq_2 | 15       | 0       | Banks to Refresh for RLD3 in sequence 2. Must not be more than 4 banks. | Read/Write |
| cfg_rld3_refresh_seq_3 | 31       | 16      | Banks to Refresh for RLD3 in sequence 3. Must not be more than 4 banks. | Read/Write |

#### 3.23.16. sbcfg2: Sideband Configuration

**address=25(32 bit)**

| Field                    | Bit High | Bit Low | Description  | Access     |
|--------------------------|----------|---------|--|------------|
| cfg_srf_zqcal_disable    | 0        | 0       | Set to 1 to disable ZQ Calibration after self refresh.   | Read/Write |
| cfg_mps_zqcal_disable    | 1        | 1       | Set to 1 to disable ZQ Calibration after Maximum Power Saving exit.  | Read/Write |
| cfg_mps_dqstrk_disable   | 2        | 2       | Set to 1 to disable DQS Tracking after Maximum Power Saving exit.  | Read/Write |
| cfg_sb_cg_disable        | 3        | 3       | Set to 1 to disable mem_ck gating during self refresh and deep power down. Clock gating is not supported when the Ping Pong PHY feature is enabled. Do not enable clock gating for Ping Pong PHY interfaces. | Read/Write |
| cfg_user_rfsh_en         | 4        | 4       | Setting to 1 to enable user refresh.   | Read/Write |
| cfg_srf_autoexit_en      | 5        | 5       | Setting to 1 to enable controller to exit Self Refresh when new command is detected.   | Read/Write |
| cfg_srf_entry_exit_block | 7        | 6       | Blocking arbiter from issuing cmds for the 4 cases, 2  | Read/Write |

**3.23.17. sbcfg3: Sideband Configuration****address=26(32 bit)**

| Field           | Bit High | Bit Low | Description                                | Access     |
|-----------------|----------|---------|--|------------|
| cfg_sb_ddr4_mr3 | 19       | 0       | This register stores the DDR4 MR3 Content. | Read/Write |

**3.23.18. sbcfg4: Sideband Configuration****address=27(32 bit)**

| Field           | Bit High | Bit Low | Description                                | Access     |
|-----------------|----------|---------|--|------------|
| cfg_sb_ddr4_mr4 | 19       | 0       | This register stores the DDR4 MR4 Content. | Read/Write |

**3.23.19. sbcfg5: Sideband Configuration**



**address=28(32 bit)**

| Field                     | Bit High | Bit Low | Description  | Access     |
|---------------------------|----------|---------|--|------------|
| cfg_short_dqstrk_ctrl_en  | 0        | 0       | Set to 1 to enable controller controlled DQS short tracking, Set to 0 to enable sequencer controlled DQS short tracking. | Read/Write |
| cfg_period_dqstrk_ctrl_en | 1        | 1       | Set to 1 to enable controller to issue periodic DQS tracking.  | Read/Write |

### 3.23.20. sbcfg6: Sideband Configuration

**address=29(32 bit)**

| Field                            | Bit High | Bit Low | Description   | Access     |
|----------------------------------|----------|---------|---|------------|
| cfg_period_dqstrk_interval       | 15       | 0       | Interval between two controller controlled periodic DQS tracking. | Read/Write |
| cfg_t_param_dqstrk_to_valid_last | 23       | 16      | DQS Tracking Rd to Valid timing for the last Rank.                | Read/Write |
| cfg_t_param_dqstrk_to_valid      | 31       | 24      | DQS Tracking Rd to Valid timing for Ranks other than the Last.    | Read/Write |

### 3.23.21. sbcfg7: Sideband Configuration

**address=30(32 bit)**

| Field                   | Bit High | Bit Low | Description  | Access     |
|-------------------------|----------|---------|--|------------|
| cfg_rfsh_warn_threshold | 6        | 0       | Threshold to warn a refresh is needed within the number of controller clock cycles specified by the threshold. | Read/Write |

### 3.23.22. sbcfg8: Sideband Configuration

**address=7(32 bit)**

| Field                    | Bit High | Bit Low | Description   | Access     |
|--------------------------|----------|---------|---|------------|
| cfg_reserve              | 15       | 1       | General purpose reserved register.                                  | Read/Write |
| cfg_ddr4_mps_addr_mirror | 0        | 0       | When asserted, indicates DDR4 Address Mirroring is enabled for MPS. | Read/Write |

### 3.23.23. sbcfg9: Sideband Configuration

**address=8(32 bit)**

| Field           | Bit High | Bit Low | Description           | Access     |
|-----------------|----------|---------|-----------------------|------------|
| cfg_sb_ddr4_mr5 | 15       | 0       | DDR4 Mode Register 5. | Read/Write |



### 3.23.24. caltiming0: Command/Address/Latency Parameters

address=31(32 bit)

| Field                            | Bit High | Bit Low | Description   | Access     |
|----------------------------------|----------|---------|---|------------|
| cfg_t_param_act_to_rdwr          | 5        | 0       | Activate to Read/write command timing.                                  | Read/Write |
| cfg_t_param_act_to_pch           | 11       | 6       | Active to precharge.  | Read/Write |
| cfg_t_param_act_to_act           | 17       | 12      | Active to activate timing on same bank.                                 | Read/Write |
| cfg_t_param_act_to_act_diff_bank | 23       | 18      | Active to activate timing on different banks, for DDR4 same bank group. | Read/Write |
| cfg_t_param_act_to_act_diff_bg   | 29       | 24      | Active to activate timing on different bank groups, DDR4 only.          | Read/Write |

### 3.23.25. caltiming1: Command/Address/Latency Parameters

address=32(32 bit)

| Field                          | Bit High | Bit Low | Description                                     | Access     |
|--------------------------------|----------|---------|---|------------|
| cfg_t_param_rd_to_rd           | 5        | 0       | Read to read command timing on same bank.       | Read/Write |
| cfg_t_param_rd_to_rd_diff_chip | 11       | 6       | Read to read command timing on different chips. | Read/Write |
| cfg_t_param_rd_to_rd_diff_bg   | 17       | 12      | Read to read command timing on different chips. | Read/Write |
| cfg_t_param_rd_to_wr           | 23       | 18      | Write to read command timing on same bank.      | Read/Write |
| cfg_t_param_rd_to_wr_diff_chip | 29       | 24      | Read to write command timing on different chips | Read/Write |

### 3.23.26. caltiming2: Command/Address/Latency Parameters

address=33(32 bit)

| Field                        | Bit High | Bit Low | Description  | Access     |
|------------------------------|----------|---------|--|------------|
| cfg_t_param_rd_to_wr_diff_bg | 5        | 0       | Read to write command timing on different bank groups. | Read/Write |
| cfg_t_param_rd_to_pch        | 11       | 6       | Read to precharge command timing.                      | Read/Write |

*continued...*



| Field                          | Bit High | Bit Low | Description   | Access     |
|--------------------------------|----------|---------|---|------------|
| cfg_t_param_rd_ap_to_valid     | 17       | 12      | Read command with autoprecharge to data valid timing. | Read/Write |
| cfg_t_param_wr_to_wr           | 23       | 18      | Write to write command timing on same bank.           | Read/Write |
| cfg_t_param_wr_to_wr_diff_chip | 29       | 24      | Write to write command timing on different chips.     | Read/Write |

### 3.23.27. caltiming3: Command/Address/Latency Parameters

address=34(32 bit)

| Field                          | Bit High | Bit Low | Description   | Access     |
|--------------------------------|----------|---------|---|------------|
| cfg_t_param_wr_to_wr_diff_bg   | 5        | 0       | Write to write command timing on different bank groups. | Read/Write |
| cfg_t_param_wr_to_rd           | 11       | 6       | Write to read command timing.                           | Read/Write |
| cfg_t_param_wr_to_rd_diff_chip | 17       | 12      | Write to read command timing on different chips.        | Read/Write |
| cfg_t_param_wr_to_rd_diff_bg   | 23       | 18      | Write to read command timing on different bank groups.  | Read/Write |
| cfg_t_param_wr_to_pch          | 29       | 24      | Write to precharge command timing.                      | Read/Write |

### 3.23.28. caltiming4: Command/Address/Latency Parameters

address=35(32 bit)

| Field                        | Bit High | Bit Low | Description   | Access     |
|------------------------------|----------|---------|---|------------|
| cfg_t_param_wr_ap_to_valid   | 5        | 0       | Write with autoprecharge to valid command timing.               | Read/Write |
| cfg_t_param_pch_to_valid     | 11       | 6       | Precharge to valid command timing.                              | Read/Write |
| cfg_t_param_pch_all_to_valid | 17       | 12      | Precharge all to banks being ready for bank activation command. | Read/Write |
| cfg_t_param_arf_to_valid     | 25       | 18      | Auto Refresh to valid DRAM command window.                      | Read/Write |
| cfg_t_param_pdn_to_valid     | 31       | 26      | Power down to valid bank command window.                        | Read/Write |

### 3.23.29. caltiming5: Command/Address/Latency Parameters

**address=36(32 bit)**

| Field                     | Bit High | Bit Low | Description                                | Access     |
|---------------------------|----------|---------|--|------------|
| cfg_t_param_srf_to_valid  | 9        | 0       | Self-refresh to valid bank command window. | Read/Write |
| cfg_t_param_srf_to_zq_cal | 19       | 10      | Self refresh to ZQ calibration window.     | Read/Write |

**3.23.30. caltiming6: Command/Address/Latency Parameters****address=37(32 bit)**

| Field                  | Bit High | Bit Low | Description                       | Access     |
|------------------------|----------|---------|-----------------------------------|------------|
| cfg_t_param_arf_period | 12       | 0       | Auto-refresh period.              | Read/Write |
| cfg_t_param_pdn_period | 28       | 13      | Clock power down recovery period. | Read/Write |

**3.23.31. caltiming7: Command/Address/Latency Parameters****address=38(32 bit)**

| Field                     | Bit High | Bit Low | Description  | Access     |
|---------------------------|----------|---------|--|------------|
| cfg_t_param_zqcl_to_valid | 8        | 0       | Long ZQ calibration to valid.  | Read/Write |
| cfg_t_param_zqcs_to_valid | 15       | 9       | Short ZQ calibration to valid.                                       | Read/Write |
| cfg_t_param_mrs_to_valid  | 19       | 16      | Mode Register Setting to valid.                                      | Read/Write |
| cfg_t_param_mps_to_valid  | 29       | 20      | Timing parameter for Maximum Power Saving to any valid command. tMPX | Read/Write |

**3.23.32. caltiming8: Command/Address/Latency Parameters****address=39(32 bit)**

| Field                          | Bit High | Bit Low | Description   | Access     |
|--------------------------------|----------|---------|---|------------|
| cfg_t_param_mrr_to_valid       | 3        | 0       | Timing parameter for Mode Register Read to any valid command.   | Read/Write |
| cfg_t_param_mpr_to_valid       | 8        | 4       | Timing parameter for Multi Purpose Register Read to any valid command.  | Read/Write |
| cfg_t_param_mps_exit_cs_to_cke | 12       | 9       | Timing parameter for exit Maximum Power Saving. Timing requirement for CS assertion vs CKE de-assertion. tMPX_S | Read/Write |

*continued...*



| Field                                | Bit High | Bit Low | Description   | Access     |
|--------------------------------------|----------|---------|---|------------|
| cfg_t_param_mps_exit_cke_to_cs       | 16       | 13      | Timing parameter for exit Maximum Power Saving. Timing requirement for CKE de-assertion vs CS de-assertion. tMPX_LH | Read/Write |
| cfg_t_param_rld3_multibank_ref_delay | 19       | 17      | RLD3 Refresh to Refresh Delay for all sequences.  | Read/Write |
| cfg_t_param_mmr_cmd_to_valid         | 27       | 20      | MMR cmd to valid delay.   | Read/Write |

### 3.23.33. caltiming9: Command/Address/Latency Parameters

address=40(32 bit)

| Field                    | Bit High | Bit Low | Description                                | Access     |
|--------------------------|----------|---------|--|------------|
| cfg_t_param_4_act_to_act | 7        | 0       | The four-activate window timing parameter. | Read/Write |

### 3.23.34. caltiming10: Command/Address/Latency Parameters

address=41(32 bit)

| Field                     | Bit High | Bit Low | Description                                     | Access     |
|---------------------------|----------|---------|---|------------|
| cfg_t_param_16_act_to_act | 7        | 0       | The 16-activate window timing parameter (RLD3). | Read/Write |

### 3.23.35. dramaddrw: Row/Column/Bank Address Width Configuration

address=42(32 bit)

| Field                     | Bit High | Bit Low | Description   | Access     |
|---------------------------|----------|---------|---|------------|
| cfg_col_addr_width        | 4        | 0       | The number of column address bits for the memory devices in your memory interface.      | Read/Write |
| cfg_row_addr_width        | 9        | 5       | The number of row address bits for the memory devices in your memory interface.         | Read/Write |
| cfg_bank_addr_width       | 13       | 10      | The number of bank address bits for the memory devices in your memory interface.        | Read/Write |
| cfg_bank_group_addr_width | 15       | 14      | The number of bank group address bits for the memory devices in your memory interface.  | Read/Write |
| cfg_cs_addr_width         | 18       | 16      | The number of chip select address bits for the memory devices in your memory interface. | Read/Write |



### 3.23.36. sideband0: Sideband

**address=43(32 bit)**

| Field          | Bit High | Bit Low | Description   | Access     |
|----------------|----------|---------|---|------------|
| mr_cmd_trigger | 0        | 0       | Write to 1 to trigger the execution of the mode register command. | Read/Write |

### 3.23.37. sideband1: Sideband

**address=44(32 bit)**

| Field           | Bit High | Bit Low | Description   | Access     |
|-----------------|----------|---------|---|------------|
| mmr_refresh_req | 3        | 0       | When asserted, indicates Refresh request to the specific rank. Each bit corresponds to each rank. | Read/Write |

### 3.23.38. sideband2: Sideband

**address=45(32 bit)**

| Field              | Bit High | Bit Low | Description  | Access     |
|--------------------|----------|---------|--|------------|
| mmr_zqcal_long_req | 0        | 0       | When asserted, indicates long ZQ cal request. This bit is write clear. | Read/Write |

### 3.23.39. sideband3: Sideband

**address=46(32 bit)**

| Field               | Bit High | Bit Low | Description   | Access     |
|---------------------|----------|---------|---|------------|
| mmr_zqcal_short_req | 0        | 0       | When asserted, indicates short ZQ cal request. This bit is write clear. | Read/Write |

### 3.23.40. sideband4: Sideband

**address=47(32 bit)**

| Field             | Bit High | Bit Low | Description  | Access     |
|-------------------|----------|---------|--|------------|
| mmr_self_rfsh_req | 3        | 0       | When asserted, indicates self refresh request to the specific rank. Each bit corresponds to each rank. These bits are write clear. | Read/Write |



### 3.23.41. sideband5: Sideband

address=48(32 bit)

| Field           | Bit High | Bit Low | Description  | Access     |
|-----------------|----------|---------|--|------------|
| mmr_dpd_mps_req | 0        | 0       | When asserted, indicates deep power down or max power saving request. This bit is write clear. | Read/Write |

### 3.23.42. sideband6: Sideband

address=49(32 bit)

| Field      | Bit High | Bit Low | Description                           | Access |
|------------|----------|---------|---------------------------------------|--------|
| mr_cmd_ack | 0        | 0       | Acknowledge to mode register command. | Read   |

### 3.23.43. sideband7: Sideband

address=50(32 bit)

| Field           | Bit High | Bit Low | Description                                     | Access |
|-----------------|----------|---------|---|--------|
| mmr_refresh_ack | 0        | 0       | Acknowledge to indicate refresh is in progress. | Read   |

### 3.23.44. sideband8: Sideband

address=51(32 bit)

| Field         | Bit High | Bit Low | Description                                   | Access |
|---------------|----------|---------|---|--------|
| mmr_zqcal_ack | 0        | 0       | Acknowledge to indicate ZQCAL is progressing. | Read   |

### 3.23.45. sideband9: Sideband

address=52(32 bit)

| Field             | Bit High | Bit Low | Description  | Access |
|-------------------|----------|---------|--|--------|
| mmr_self_rfsh_ack | 0        | 0       | Acknowledge to indicate self refresh is progressing. | Read   |

### 3.23.46. sideband10: Sideband

**address=53(32 bit)**

| Field           | Bit High | Bit Low | Description  | Access |
|-----------------|----------|---------|--|--------|
| mmr_dpd_mps_ack | 0        | 0       | Acknowledge to indicate deep power down/max power saving is in progress. | Read   |

**3.23.47. sideband11: Sideband****address=54(32 bit)**

| Field           | Bit High | Bit Low | Description   | Access |
|-----------------|----------|---------|---|--------|
| mmr_auto_pd_ack | 0        | 0       | Acknowledge to indicate auto power down is in progress. | Read   |

**3.23.48. sideband12: Sideband****address=55(32 bit)**

| Field       | Bit High | Bit Low | Description  | Access     |
|-------------|----------|---------|--|------------|
| mr_cmd_type | 2        | 0       | Indicates the type of Mode Register Command.                   | Read/Write |
| mr_cmd_rank | 6        | 3       | Indicates which rank the mode register command is intended to. | Read/Write |

**3.23.49. sideband13: Sideband****address=56(32 bit)**

| Field         | Bit High | Bit Low | Description   | Access     |
|---------------|----------|---------|---|------------|
| mr_cmd_opcode | 31       | 0       | [31:27] reserved. Register Command Opcode Information to be used for Register Command LPDDR3 [26:20]<br>Reserved [19:10] falling edge CA.<br>[9:0] rising edge CA DDR4 [26:24]<br>C2:C0 [23] ACT [22:21] BG1:BG0<br>[20] Reserved [19:18] BA1:BA0 [17]<br>A17 [16] RAS [15] CAS [14] WE<br>[13:0] A13:A0 DDR3 [26:21]<br>Reserved [20:18] BA2:BA0 [17] A17<br>[16] RAS [15] CAS [14] WE [13]<br>Reserved [12:0] A12:A0 RLDRAM3<br>[26] Reserved [25:22] BA3:BA0 [21]<br>REF [20] WE [19:0] A19:A0 | Read/Write |

**3.23.50. sideband14: Sideband**



#### address=57(32 bit)

| Field            | Bit High | Bit Low | Description  | Access     |
|------------------|----------|---------|--|------------|
| mmr_refresh_bank | 15       | 0       | User refresh bank information, binary representation of bank address. Enables refresh to that bank address when requested. | Read/Write |

#### 3.23.51. sideband15: Sideband

#### address=58(32 bit)

| Field          | Bit High | Bit Low | Description                                   | Access     |
|----------------|----------|---------|---|------------|
| mmr_stall_rank | 3        | 0       | Setting to 1 to stall the corresponding rank. | Read/Write |

#### 3.23.52. dramsts: Calibration Status

#### address=59(32 bit)

| Field           | Bit High | Bit Low | Description  | Access |
|-----------------|----------|---------|--|--------|
| phy_cal_success | 0        | 0       | This bit will be set to 1 if the PHY was able to successfully calibrate. | Read   |
| phy_cal_fail    | 1        | 1       | This bit will be set to 1 if the PHY was unable to calibrate.            | Read   |

#### 3.23.53. ecc1: ECC General Configuration

#### address=128(32 bit)

| Field               | Bit High | Bit Low | Description  | Access     |
|---------------------|----------|---------|--|------------|
| cfg_enable_ecc      | 0        | 0       | A value of 1 enables the ECC encoder/decoder.  | Read/Write |
| cfg_enable_dm       | 1        | 1       | A value of 1 indicate that this is a design with DM.   | Read/Write |
| cfg_enable_rmw      | 2        | 2       | A value of 1 enables the RMW feature, including partial/dummy write support.                               | Read/Write |
| cfg_data_rate       | 6        | 3       | Set this value to 2, 4, or 8 for full, half or quarter rate designs.                                       | Read       |
| cfg_ecc_in_protocol | 7        | 7       | Set this value to 1 for Avalon-MM or 0 for Avalon-ST input interface. Read-only register.                  | Read/Write |
| cfg_enable_auto_cor | 8        | 8       | A value of 1 enables the auto correction feature, injecting a dummy write command after a single-bit error | Read/Write |

*continued...*



| Field                         | Bit High | Bit Low | Description  | Access     |
|-------------------------------|----------|---------|--|------------|
|                               |          |         | is (SBE) detected. This feature must be enabled together with RMW and ECC.   |            |
| cfg_enable_ecc_code_overwrite | 9        | 9       | A value of 1 enables the ECC code overwrite feature. Reuse the original read-back ECC code during RMW if a double-bit error (DBE) is detected. | Read/Write |
| Reserved                      | 31       | 10      |  |            |

### 3.23.54. ecc2: Width Configuration

address=129(32 bit)

| Field                | Bit High | Bit Low | Description   | Access |
|----------------------|----------|---------|---|--------|
| cfg_dram_data_width  | 7        | 0       | Set this value to the DRAM data width, without taking rate conversion into consideration. For example, for a $64+8=72$ bit ECC design, set this register to 'd72'.                    | Read   |
| cfg_local_data_width | 15       | 8       | Set this value to the LOCAL data width, without taking rate conversion into consideration. For example, for a $64+8=72$ DQ bit ECC design, set this register to 'd64'.                | Read   |
| cfg_addr_width       | 21       | 16      | Set this value to the LOCAL address width, which corresponds to the address field width of the cmd field. For example, for a LOCAL address width of 24 bits, set this value to 'd24'. | Read   |
| Reserved             | 31       | 22      |   |        |

### 3.23.55. ecc3: ECC Error and Interrupt Configuration

address=130(32 bit)

| Field             | Bit High | Bit Low | Description  | Access     |
|-------------------|----------|---------|--|------------|
| cfg_gen_sbe       | 0        | 0       | A value of 1 enables the generate SBE feature. Generates a single bit error during the write process.                                  | Read/Write |
| cfg_gen_dbe       | 1        | 1       | A value of 1 enables the generate DBE feature. Generates a double bit error during the write process.                                  | Read/Write |
| cfg_enable_intr   | 2        | 2       | A value of 1 enables the interrupt feature. The interrupt signal notifies if an error condition occurs. The condition is configurable. | Read/Write |
| cfg_mask_sbe_intr | 3        | 3       | A value of 1 masks the interrupt signal when SBE occurs.   | Read/Write |

*continued...*



| Field                       | Bit High | Bit Low | Description  | Access     |
|-----------------------------|----------|---------|--|------------|
| cfg_mask_dbe_intr           | 4        | 4       | A value of 1 masks the interrupt signal when DBE occurs.   | Read/Write |
| cfg_mask_corr_drop_ped_intr | 5        | 5       | A value of 1 masks the interrupt signal when auto correction command can't be scheduled, due to back-pressure (FIFO full). | Read/Write |
| cfg_mask_hmi_intr           | 6        | 6       | A value of 1 masks the interrupt signal when the hard memory interface asserts an interrupt signal via hmi_interrupt port. | Read/Write |
| cfg_clr_intr                | 7        | 7       | Writing a value of 1 to this self-clearing bit clears the interrupt signal, error status, and address.                     | Read/Write |
| Reserved                    | 31       | 9       |  |            |

### 3.23.56. ecc4: Status and Error Information

**address=144(32 bit)**

| Field                  | Bit High | Bit Low | Description  | Access |
|------------------------|----------|---------|--|--------|
| sts_ecc_intr           | 0        | 9       | Indicates the interrupt status; a value of 1 indicates interrupt occurred.                             | Read   |
| sts_sbe_error          | 1        | 1       | Indicates the SBE status; a value of 1 indicates SBE occurred.   | Read   |
| sts_dbe_error          | 2        | 2       | Indicates the DBE status; a value of 1 indicates DBE occurred.   | Read   |
| sts_corr_dropped       | 3        | 3       | Indicates the status of correction command dropped; a value of 1 indicates correction command dropped. | Read   |
| sts_sbe_count          | 7        | 4       | Indicates the number of times SBE error has occurred. The counter will overflow.                       | Read   |
| sts_dbe_count          | 11       | 8       | Indicates the number of times DBE error has occurred. The counter will overflow.                       | Read   |
| sts_corr_dropped_count | 15       | 12      | Indicates the number of times correction command has dropped. Counter will overflow.                   | Read   |
| Reserved               | 31       | 16      |  |        |

### 3.23.57. ecc5: Address of Most Recent SBE/DBE

**address=145(32 bit)**

| Field         | Bit High | Bit Low | Description  | Access |
|---------------|----------|---------|--|--------|
| sts_err_addr* | 31       | 0       | Address of the most recent single-bit error or double-bit error. | Read   |

**3.23.58. ecc6: Address of Most Recent Correct Command Dropped****address=146(32 bit)**

| Field                 | Bit High | Bit Low | Description  | Access |
|-----------------------|----------|---------|--|--------|
| sts_corr_dropped_addr | 31       | 0       | Address of the most recent correction command dropped. | Read   |

**3.24. Document Revision History**

| Date          | Version    | Changes  |
|---------------|------------|--|
| May 2017      | 2017.05.08 | <ul style="list-style-type: none"><li>Added <i>Using the EMIF Debug Toolkit with Arria 10 HPS Interfaces</i> topic.</li><li>Rebranded as Intel.</li></ul>  |
| October 2016  | 2016.10.31 | <ul style="list-style-type: none"><li>Added <i>Back-to-Back User-Controlled Refresh for Hard Memory Controller</i> topic.</li><li>Modified first bullet point in the <i>ECC in Arria 10 EMIF IP</i> topic.</li><li>Added additional content about read-modify-write operations to the <i>ECC in Arria 10 EMIF IP</i> topic.</li><li>Removed afi_alert_n from the <i>AFI Address and Command Signals</i> table in the <i>AFI Address and Command Signals</i> topic.</li><li>Added <i>ecc5: Address of Most Recent SBE/DBE</i> and <i>ecc6: Address of Most recent Correct Command Dropped to Memory Mapped Register (MMR) Tables</i> section.</li></ul>   |
| May 2016      | 2016.05.02 | <ul style="list-style-type: none"><li>Modified text of <i>Hard Memory Controller and Hard PHY</i> and <i>Soft Memory Controller and Hard PHY</i>.</li><li>Modified content of the <i>Ping Pong PHY Architecture</i> topic.</li><li>Added section on read-modify-write operations, to <i>ECC in Arria 10 EMIF IP</i> topic.</li><li>Added <i>AFI 4.0 Timing Diagrams</i> section.</li><li>Added <i>Arria 10 EMIF Latency</i> section.</li><li>Added <i>Arria 10 EMIF Calibration Times</i> section.</li><li>Added <i>Integrating a Custom Controller with the Hard PHY</i> section.</li></ul>   |
| November 2015 | 2015.11.02 | <ul style="list-style-type: none"><li>Added LPDDR3 support to <i>AFI Parameters</i>, <i>AFI Address and Command Signals</i>, and <i>AFI Write Data Signals</i>.</li><li>Revised rendering of address information in the <i>Memory Mapped Register (MMR) Tables</i>.</li><li>Added <i>ecc1: ECC General Configuration</i>, <i>ecc2: Width Configuration</i>, <i>ecc3: ECC Error and Interrupt Configuration</i>, and <i>ecc4: Status and Error Information</i> in the <i>Memory Mapped Register (MMR) Tables</i>.</li><li>Removed <i>RZQ Pin Sharing</i> section.</li><li>Added bank numbers to figure in <i>Restrictions on I/O Bank Usage for Arria 10 EMIF IP with HPS</i> topic.</li><li>Added <i>Arria 10 EMIF and SmartVID</i> topic.</li><li>Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i>.</li></ul> |

**continued...**

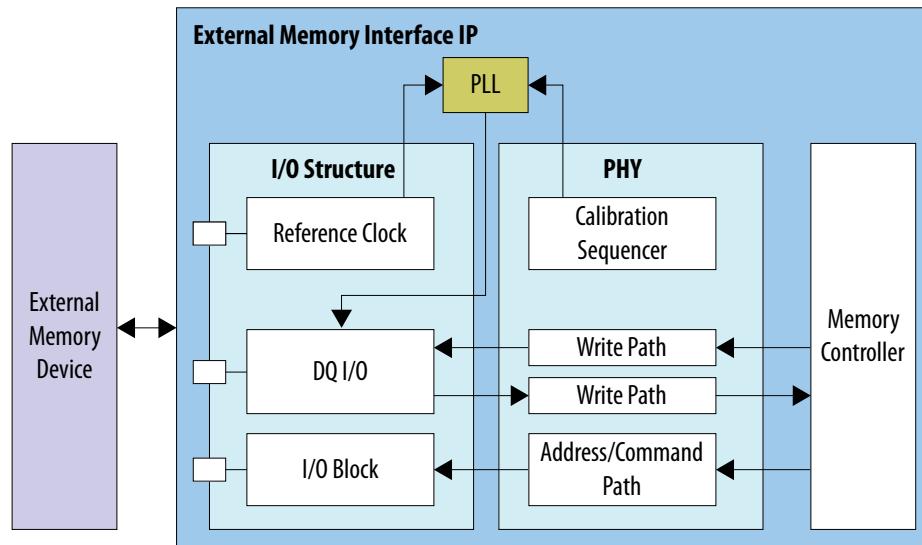


| Date          | Version    | Changes   |
|---------------|------------|---|
| May 2015      | 2015.05.04 | Maintenance release.  |
| December 2014 | 2014.12.15 | <ul style="list-style-type: none"><li>Added debug-related connections to the <i>Logical Connections</i> table in the <i>Logical Connections</i> topic.</li><li>Added Arria 10 EMIF Ping Pong PHY section.</li><li>Added Arria 10 EMIF Debugging Examples section.</li><li>Added x4 mode support.</li><li>Added AFI 4.0 Specification section.</li><li>Added MMR Register Tables section.</li></ul>  |
| August 2014   | 2014.08.15 | <ul style="list-style-type: none"><li>Added PHY-only support for DDR3, DDR4, and RLDRAM 3, and soft controller and hard PHY support for RLDRAM 3 and QDR II/II+/II+ Xtreme to <i>Supported Memory Protocols</i> table.</li><li>Added afi_conduit_end, afi_clk_conduit_end, afi_half_clk_conduit_end, and afi_reset_n_conduit_end to <i>Logical Connections</i> table.</li><li>Expanded description of ctrl1_amm_avalon_slave in <i>Logical Connections</i> table.</li><li>Added ECC in Arria 10 EMIF IP.</li><li>Added Configuring Your EMIF IP for Use with the Debug Toolkit.</li><li>Added Arria 10 EMIF for Hard Processor Subsystem.</li></ul> |
| December 2013 | 2013.12.16 | Initial release.  |

## 4. Functional Description—Intel MAX® 10 EMIF IP

Intel MAX® 10 FPGAs provide advanced processing capabilities in a low-cost, instant-on, small-form-factor device featuring capabilities such as digital signal processing, analog functionality, Nios II embedded processor support and memory controllers.

**Figure 135. MAX 10 EMIF Block Diagram**

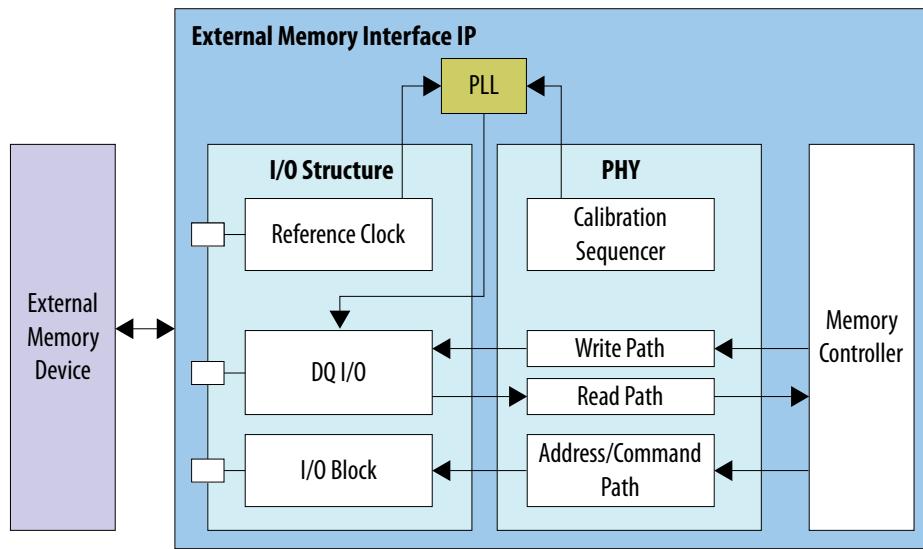


### 4.1. MAX 10 EMIF Overview

MAX 10 FPGAs provide advanced processing capabilities in a low-cost, instant-on, small-form-factor device featuring capabilities such as digital signal processing, analog functionality, Nios II embedded processor support and memory controllers.



**Figure 136. MAX 10 EMIF Block Diagram**



## 4.2. External Memory Protocol Support

MAX 10 FPGAs offer external memory interface support for DDR2, DDR3, DDR3L, and LPDDR2 protocols.

**Table 59. Supported External Memory Configurations**

| External Memory Protocol | Maximum Frequency      | Configuration                                       |
|--------------------------|------------------------|---|
| DDR3, DDR3L              | 303 MHz                | x16 + ECC + Configuration and Status Register (CSR) |
| DDR2                     | 200 MHz                | x16 + ECC + Configuration and Status Register (CSR) |
| LPDDR2 <sup>(1)</sup>    | 200 MHz <sup>(2)</sup> | x16   |

## 4.3. MAX 10 Memory Controller

MAX 10 FPGAs use the HPC II external memory controller.

## 4.4. MAX 10 Low Power Feature

The MAX 10 low power feature is automatically activated when the self refresh or low power down modes are activated. The low power feature sends the `afi_mem_clk_disable` signal to stop the clock used by the controller.

<sup>(1)</sup> MAX 10 devices support only single-die LPDDR2.

<sup>(2)</sup> To achieve the specified performance, constrain the memory device I/O and core power supply variation to within ±3%. By default, the frequency is 167 MHz.

To conserve power, the MAX 10 UniPHY IP core performs the following functions:

- Tri-states the address and command signals except CKE and RESET\_N signals
- Disables the input buffer of DDR input

**Note:** The MAX 10 low power feature is available from version 15.0 of the Quartus Prime software. To enable this feature, regenerate your MAX 10 UniPHY IP core using the Quartus Prime software version 15.0 or later.

## 4.5. MAX 10 Memory PHY

MAX 10 devices employ UniPHY, but without using DLLs for calibration. The physical layer implementation for MAX 10 external memory interfaces provides a calibrated read path and a static write path.

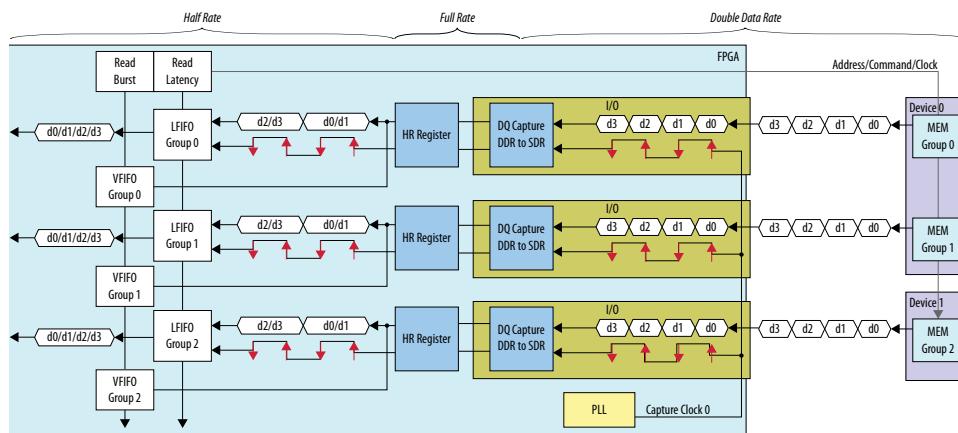
### 4.5.1. Supported Topologies

The memory PHY supports DDR2 and DDR3 protocols with up to two discrete memory devices, and the LPDDR2 protocol with one discrete memory device.

### 4.5.2. Read Datapath

One PLL output is used to capture data from the memory during read operations. The clock phase is calibrated by the sequencer before the interface is ready for use.

**Figure 137. Read Datapath**

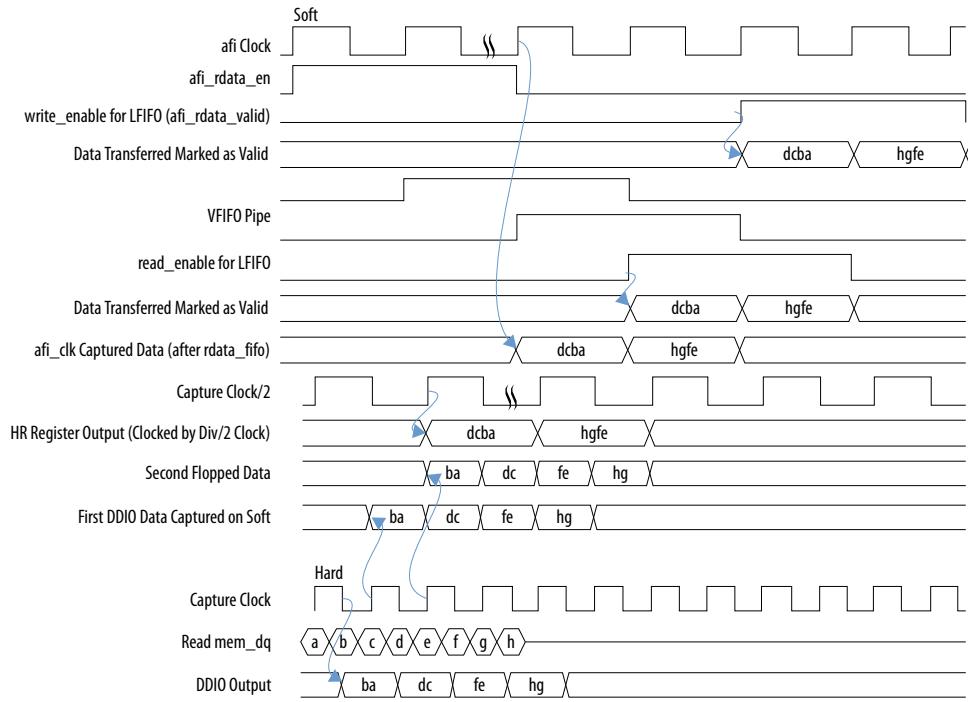


For DDR3 interfaces, two PLL outputs capture data from the memory devices during a read. In a 24-bit interface—whether the top 8 bits are used by ECC or not—the supported topology is two discrete DDR3 devices of 16-bit and 8-bit DQ each. Each discrete device has a dedicated capture clock output from the PLL.

For LPDDR2 interfaces, the supported configuration is a single memory device with memory width of 16-bit DQ. The other PLL output is used for DQS tracking purposes, because the tDQSCK drift might cause data capture to fail otherwise. The tracking clock is a shifted capture clock used to sample the DQS signal. By capturing the DQS signal, the system can compensate for DQS signal drift.



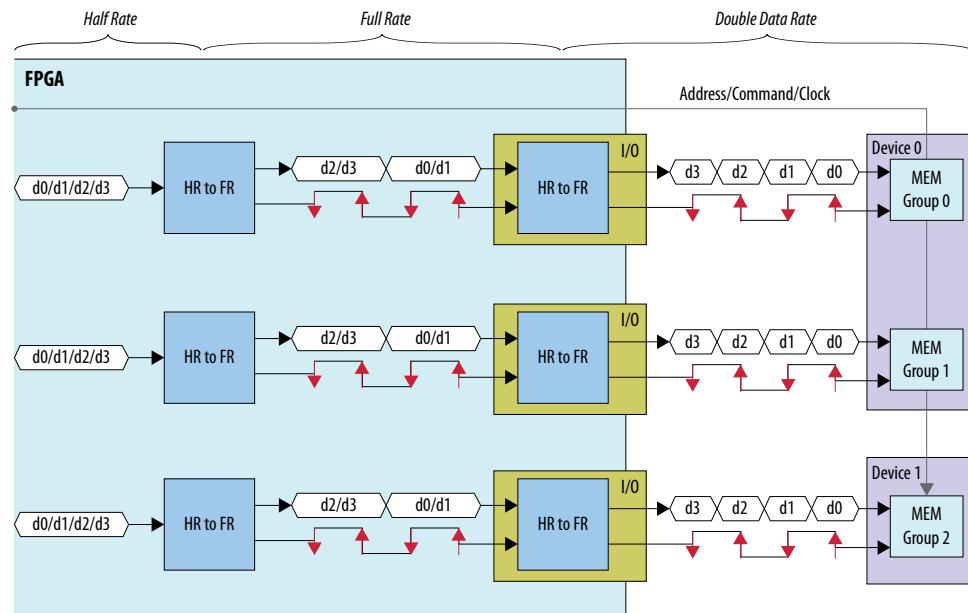
**Figure 138. Read Datapath Timing Diagram**



#### 4.5.3. Write Datapath

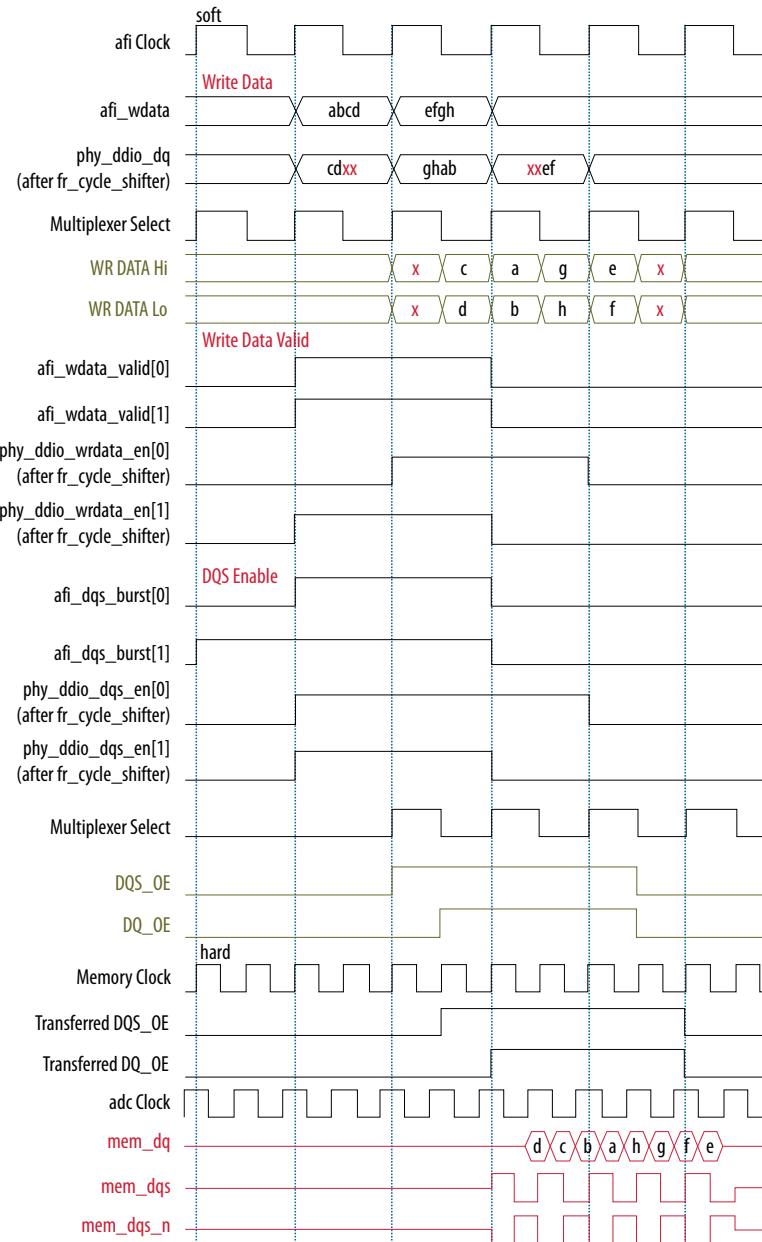
The write datapath is a static path and is timing analyzed to meet timing requirements.

**Figure 139. Write Datapath**



In the PHY write data path, for even write latency the write data valid, write data, and dqs enable pass through one stage of fr\_cycle\_shifter in a flow through path. For odd memory write latency, the output is shifted by a full-rate clock cycle. The full-rate cycle-shifted output feeds into a simple DDIO.

**Figure 140. Write Datapath Timing Diagram**





#### 4.5.4. Address and Command Datapath

Implementation of the address and command datapath differs between DDR2/DDR3 and LPDDR2, because of the double data-rate command in LPDDR2.

##### LPDDR2

For LPDDR2, CA is four bits wide on the AFI interface. The least significant bit of CA is captured by a negative-edge triggered flop, and the most-significant bit of CA is captured by a positive-edge triggered flop, before multiplexing to provide maximum setup and hold margins for the AFI clock-to-MEM clock data transfer.

##### DDR2/DDR3

For DDR2/DDR3, the full-rate register and MUX architecture is similar to LPDDR2, but both phases are driven with the same signal. The DDR3 address and command signal is not clocked by the write/ADC clock as with LPDDR2, but by the inverted MEM clock, for better address and command margin.

##### Chip Select

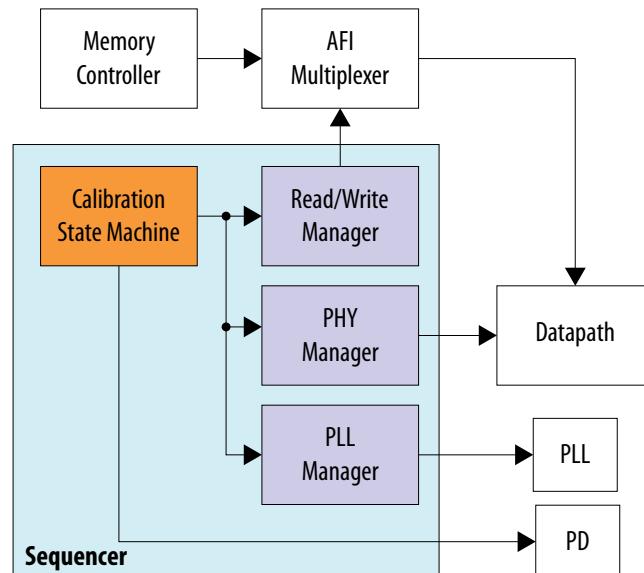
Because the memory controller can drive both phases of `cs_n` in half-rate, the signal is fully exposed to the AFI side.

#### 4.5.5. Sequencer

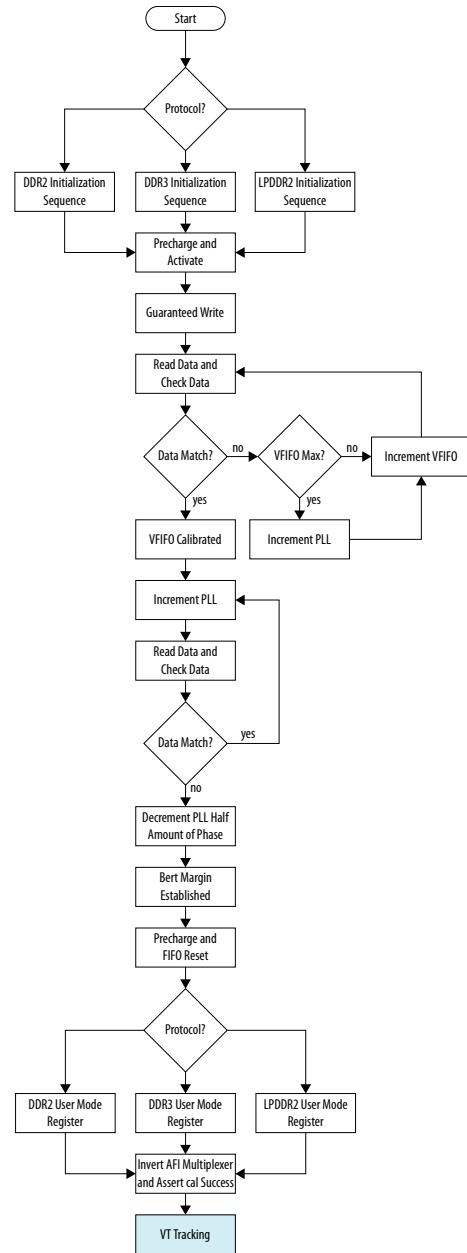
The sequencer employs an RTL-based state machine which assumes control of the interface at reset (whether at initial startup or when the IP is reset) and maintains control throughout the calibration process. The sequencer relinquishes control to the memory controller only after successful calibration.

The sequencer consists of a calibration state machine, together with a read-write (RW) manager, PHY Manager, and PLL Manager.

**Figure 141. Sequencer Architecture**



**Figure 142. Calibration State Machine Stages**





### RW Manager

The read-write (RW) manager encapsulates the protocol to read and write to the memory device through the Altera PHY Interface (AFI). It provides a buffer that stores the data to be sent to and read from memory, and provides the following commands:

- Write configuration—configures the memory for use. Sets up burst lengths, read and write latencies, and other device specific parameters.
- Refresh—initiates a refresh operation at the DRAM. The sequencer also provides a register that determines whether the RW manager automatically generates refresh signals.
- Enable or disable multi-purpose register (MPR)—for memory devices with a special register that contains calibration specific patterns that you can read, this command enables or disables access to the register.
- Activate row—for memory devices that have both rows and columns, this command activates a specific row. Subsequent reads and writes operate on this specific row.
- Precharge—closes a row before you can access a new row.
- Write or read burst—writes or reads a burst length of data.
- Write guaranteed—writes with a special mode where the memory holds address and data lines constant. Intel guarantees this type of write to work in the presence of skew, but constrains to write the same data across the entire burst length.
- Write and read back-to-back—performs back-to-back writes or reads to adjacent banks. Most memory devices have strict timing constraints on subsequent accesses to the same bank, thus back-to-back writes and reads have to reference different banks.
- Protocol-specific initialization—a protocol-specific command required by the initialization sequence.

### PHY Manager

The PHY Manager provides access to the PHY for calibration, and passes relevant calibration results to the PHY. For example, the PHYManager sets the VFIFO and LFIFO buffer parameters resulting from calibration, signals the PHY when the memory initialization sequence finishes, and reports the pass/fail status of calibration.

### PLL Manager

The PLL Manager controls the phase of capture clocks during calibration. The output phases of individual PLL outputs can be dynamically adjusted relative to each other and to the reference clock without having to load the scan chain of the PLL. The phase is shifted by 1/8th of the period of the voltage-controlled oscillator (VCO) at a time. The output clocks are active during this dynamic phase-shift process.

A PLL counter increments with every phase increase and decrements with every phase reduction. The PLL Manager records the amount by which the PLL counter has shifted since the last reset, enabling the sequencer and tracking manager to determine whether the phase boundary has been reached.



## 4.6. Calibration

The purpose of calibration is to exercise the external memory interface to find an optimal window for capture clock. There is no calibration for writing data or for address and command output; these paths are analyzed by TimeQuest to meet timing requirements.

The calibration algorithm assumes that address and command signals can reliably be sent to the external memory device, and that write data is registered with DQS.

### 4.6.1. Read Calibration

Read calibration consists of two primary parts: capture clock and VFIFO buffer calibration, and read latency tuning.

A VFIFO buffer is a FIFO buffer that is calibrated to reflect the read latency of the interface. The calibration process selects the correct read pointer of the FIFO. The VFIFO function delays the controller's `afi_rdata_valid` signal to align to data captured internally to the PHY. LFIFO buffers are FIFO buffers which ensure that data from different DQS groups arrives at the user side at the same time. Calibration ensures that data arrives at the user side with minimal latency.

#### Capture Clock and VFIFO Calibration

Capture clock and VFIFO calibration performs the following steps:

- Executes a guaranteed write routine in the read-write (RW) manager.
- Sweeps VFIFO buffer values, beginning at 0, in the PHY Manager.
  - For each adjustment of the VFIFO buffer, sweeps the capture clock phase in the PLL Manager to find the first phase that works. This is accomplished by issuing a read to the RW Manager and performing a bit check. Data is compared for all data bits.
- Increments the capture clock phase in the PLL Manager until capture clock phase values are exhausted or until the system stops working. If there are no more capture clock phase values to try, calibration increments the VFIFO buffer value in the PHY Manager, and phase sweep again until the system stops working.  
Completion of this step establishes a working range of values.
- As a final step, calibration centers the capture clock phase within the working range.

#### Read Latency Tuning

Read latency tuning performs the following steps to achieve the optimal latency value:

- Assigns one LFIFO buffer for each DQ group.
- Aligns read data to the AFI clock.
- Gradually reduces LFIFO latency until reads fail, then increases latency to find the minimum value that yields reliable operation.

### 4.6.2. Write Calibration

There is no calibration routine for writing data.



## 4.7. Sequencer Debug Information

Following calibration, the sequencer loads a set of debug information onto an output port. You can use the SignalTap II Logic Analyzer to access the debug information in the presynthesized design.

You could also bring this port to a register, to latch the value and make it accessible to a host processor.

The output port where the debug information is available is not normally connected to anything, so it could be removed during synthesis.

Signal name: phy\_cal\_debug\_info

Module: <corename>\_s0.v

The signal is 32 bits wide, and is defined in the following table.

**Table 60. Sequencer Debug Data**

|                          |   |
|--------------------------|---|
| best_comp_result [23:16] | Best data comparison result on a per-pin basis from the Read-Write Manager. Pin 16 - 23.<br>Any mismatch on respective pin data comparison produces a high bit.<br>If calibration fails, the result is a non-zero value. When calibration passes, the result is zero. |
| best_comp_result [15:8]  | Best data comparison result on a per-pin basis from the Read-Write Manager. Pin 8 - 15.<br>Any mismatch on respective pin data comparison produces a high bit.<br>If calibration fails, the result is a non-zero value. When calibration passes, the result is zero.  |
| best_comp_result [7:0]   | Best data comparison result on a per-pin basis from the Read-Write Manager. Pin 0 - 7.<br>Any mismatch on respective pin data comparison produces a high bit.<br>If calibration fails, the result is a non-zero value. When calibration passes, the result is zero.   |
| margin [7:0]             | Margin found by the sequencer if calibration passes.<br>Number represents the amount of subsequent PLL phase where valid data is found.<br>If calibration fails, this number is zero. When calibration passes, this value is non-zero.                                |

### Debug Example and Interpretation

The following table illustrates possible debug signal values and their interpretations.

|                     | <b>best_comp_result [23:16]</b> | <b>best_comp_result [15:8]</b> | <b>best_comp_result [7:0]</b> | <b>margin [7:0]</b>  |
|---------------------|---------------------------------|--------------------------------|-------------------------------|--|
| Passing result      | 0000 0000                       | 0000 0000                      | 0000 0000                     | 0001 0000  |
| Interpretation      | No failing pins.                | No failing pins.               | No failing pins.              | 16 phases of margin for valid window. Ideal case for 300MHz interface. |
| <i>continued...</i> |                                 |                                |                               |  |



|                | best_comp_result [23:16] | best_comp_result [15:8]     | best_comp_result [7:0] | margin [7:0]     |
|----------------|--------------------------|-----------------------------|------------------------|------------------|
| Failing result | 0010 0000                | 1111 1111                   | 0000 0000              | 0000 0000        |
| Interpretation | Pin 21 failing.          | All pins failing, pin 8-15. | No failing pins.       | No valid window. |

## 4.8. Register Maps

This topic provides register information for MAX 10 EMIF.

**Table 61. Controller Register Map**

| Address       | Description                |
|---------------|----------------------------|
| 0x100 - 0x126 | Reserved                   |
| 0x130         | ECC control register       |
| 0x131         | ECC status register        |
| 0x132         | ECC error address register |

UniPHY does not have a configuration and status register. The HPC II controller does have a configuration and status register, when configured for ECC.

### Related Information

[Soft Controller Register Map](#) on page 364

## 4.9. Document Revision History

**Table 62. Document Revision History**

| Date          | Version    | Changes  |
|---------------|------------|--|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release.   |
| May 2016      | 2016.05.02 | Maintenance release.   |
| November 2015 | 2015.11.02 | Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i> .   |
| May 2015      | 2015.05.04 | <ul style="list-style-type: none"><li>Updated the external memory protocol support.</li><li>Added topic about the low power feature.</li></ul> |
| December 2014 | 2014.12.15 | Initial release.   |

## 6. Functional Description—Hard Memory Interface

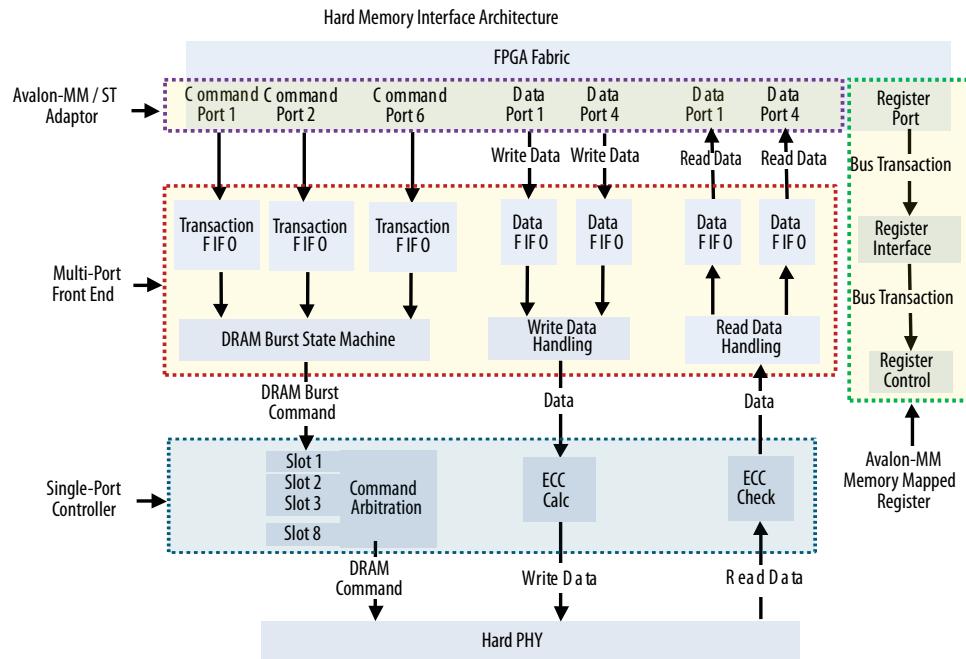
The hard (on-chip) memory interface components are available in the Arria V and Cyclone V device families.

The Arria V device family includes hard memory interface components supporting DDR2 and DDR3 SDRAM memory protocols at speeds of up to 533 MHz. For the Quartus II software version 12.0 and later, the Cyclone V device family supports both hard and soft interface support.

The Arria V device family supports both hard and soft interfaces for DDR3 and DDR2, and soft interfaces for LPDDR2 SDRAM, QDR II SRAM, and RLDRAM II memory protocols. The Cyclone V device family supports both hard and soft interfaces for DDR3, DDR2, and LPDDR2 SDRAM memory protocols.

The hard memory interface consists of three main parts, as follows:

- The multi-port front end (MPFE), which allows multiple independent accesses to the hard memory controller.
- The hard memory controller, which initializes, refreshes, manages, and communicates with the external memory device.
- The hard PHY, which provides the physical layer interface to the external memory device.

**Figure 143. Hard Memory Interface Architecture**


## 6.1. Multi-Port Front End (MPFE)

The multi-port front end and its associated fabric interface provide up to six command ports, four read-data ports and four write-data ports, through which user logic can access the hard memory controller. Each port can be configured as read only or write only, or read and write ports may be combined to form bidirectional data ports. Ports can be 32, 64, 128, or 256 data bits wide, depending on the number of ports used and the type (unidirectional or bidirectional) of the port.

### Fabric Interface

The fabric interface provides communication between the Avalon-ST-like internal protocol of the hard memory interface and the external Avalon-MM protocol. The fabric interface supports frequencies in the range of 10 MHz to one-half of the memory interface frequency. For example, for an interface running at 533 MHz, the maximum user logic frequency is 267 MHz. The MPFE handles the clock crossing between user logic and the hard memory interface.

The multi-port front end read and write FIFO depths are 8, and the command FIFO depth is 4. The FIFO depths are not configurable.

### Operation Ordering

Requests arriving at a given port are executed in the order in which they are received.

Requests arriving at different ports have no guaranteed order of service, except when a first transaction has completed before the second arrives.



## 6.2. Multi-port Scheduling

Multi-port scheduling is governed by two considerations: the absolute priority of a request and the weighting of a port. User-configurable priority and weight settings determine the absolute and relative scheduling policy for each port.

### 6.2.1. Port Scheduling

The evaluation of absolute priority ensures that ports carrying higher-priority traffic are served ahead of ports carrying lower-priority traffic. The scheduler recognizes eight priority levels, with higher values representing higher priorities. Priority is absolute; for example, any transaction with priority seven will always be scheduled before transactions of priority six or lower.

When ports carry traffic of the same absolute priority, relative priority is determined based on port weighting. Port weighting is a five-bit value, and is determined by a weighted round robin (WRR) algorithm.

The scheduler can alter priority if the latency target for a transaction is exceeded. The scheduler tracks latency on a per-port basis, and counts the cycles that a transaction is pending. Each port has a priority escalation register and a pending counter engagement register. If the number of cycles in the pending counter engagement register elapse without a pending transaction being served, that transaction's priority is escalated.

To ensure that high-priority traffic is served quickly and that long and short bursts are effectively interleaved on ports, bus transactions longer than a single DRAM burst are scheduled as a series of DRAM bursts, with each burst arbitrated separately.

The scheduler uses a form of deficit round robin (DRR) scheduling algorithm which corrects for past over-servicing or under-servicing of a port. Each port has an associated weight which is updated every cycle, with a user-configured weight added to it and the amount of traffic served subtracted from it. The port with the highest weighting is considered the most eligible.

To ensure that lower priority ports do not build up large running weights while higher priority ports monopolize bandwidth, the hard memory controller's DRR weights are updated only when a port matches the scheduled priority. Hence, if three ports have traffic, two being priority 7 and one being priority 4, the weights for both ports at priority 7 are updated but the port with priority 4 remains unchanged.

### 6.2.2. DRAM Burst Scheduling

DRAM burst scheduling recognizes addresses that access the same column/row combination—also known as open page accesses. Such operations are always served in the order in which they are received in the single-port controller.

Selection of DRAM operations is a two-stage process; first, each pending transaction must wait for its timers to be eligible for execution, then the transaction arbitrates against other transactions that are also eligible for execution.



The following rules govern transaction arbitration:

- High priority operations take precedence over lower priority operations
- If multiple operations are in arbitration, read operations have precedence over write operations
- If multiple operations still exist, the oldest is served first

A high-priority transaction in the DRAM burst scheduler wins arbitration for that bank immediately if the bank is idle and the high-priority transaction's chip select/row/column address does not match an address already in the single-port controller. If the bank is not idle, other operations to that bank yield until the high-priority operation is finished. If the address matches another chip select/row/column, the high-priority transaction yeilds until the earlier transaction is completed.

You can force the DRAM burst scheduler to serve transactions in the order that they are received, by setting a bit in the register set.

### 6.2.3. DRAM Power Saving Modes

The hard memory controller supports two DRAM power-saving modes: self-refresh, and fast/slow all-bank precharge powerdown exit. Engagement of a DRAM power saving mode can occur due to inactivity, or in response to a user command.

The user command to enter power-down mode forces the DRAM burst-scheduling bank-management logic to close all banks and issue the power-down command. You can program the controller to power down when the DRAM burst-scheduling queue is empty for a specified number of cycles; the DRAM is reactivated when an active DRAM command is received.

## 6.3. MPFE Signal Descriptions

The following table describes the signals for the multi-port front end.

**Table 63. MPFE Signals**

| Signal                         | Direction | Description  |
|--------------------------------|-----------|--|
| avl_<signal_name>_# (1)        | —         | Local interface signals.   |
| mp_cmd_clk_#_clk (2)           | Input     | Clock for the command FIFO buffer. (3) Follow Avalon-MM master frequency. Maximum frequency is one-half of the interface frequency, and subject to timing closure. |
| mp_cmd_reset_n_#_reset_n (1)   | Input     | Asynchronous reset signal for command FIFO buffer.   |
| mp_rfifo_clk_#_clk (2)         | Input     | Clock for the read data FIFO buffer. Follow Avalon-MM master frequency. Maximum frequency is one-half of the interface frequency, and subject to timing closure.   |
| mp_rfifo_reset_n_#_reset_n (2) | Input     | Asynchronous reset signal for read data FIFO buffer.   |
| mp_wfifo_clk_#_clk (2)         | Input     | Clock for the write data FIFO buffer. Follow Avalon-MM master frequency. Maximum frequency is one-half of the interface frequency, and subject to timing closure.  |
| mp_wfifo_reset_n_#_reset_n (2) | Input     | Asynchronous reset signal for write data FIFO buffer.  |

*continued...*



| Signal            | Direction | Description   |
|-------------------|-----------|---|
| bonding_in_1/2/3  | Input     | Bonding interface input port. Connect second controller bonding output port to this port according to the port sequence.      |
| bonding_out_1/2/3 | Output    | Bonding interface output port. Connect this port to the second controller bonding intput port according to the port sequence. |

Notes to Table:

1. # represents the number of the slave port. Values are 0–5.
2. # represents the number of the slave port. Values are 0–3.
3. The command FIFO buffers have two stages. The first stage is 4 bus transaction deep per port. After port scheduling, the commands are placed in the second stage FIFO buffer, which is 8 DRAM transactions deep. The second stage FIFO buffer is used to optimize memory operations where bank look ahead and data reordering occur. The write data buffer is 32 deep and read buffer is 64 deep.

Every input interface (command, read data, and write data) has its own clock domain. Each command port can be connected to a different clock, but the read data and write data ports associated with a command port must connect to the same clock as that command port. Each input interface uses the same reset signal as its clock.

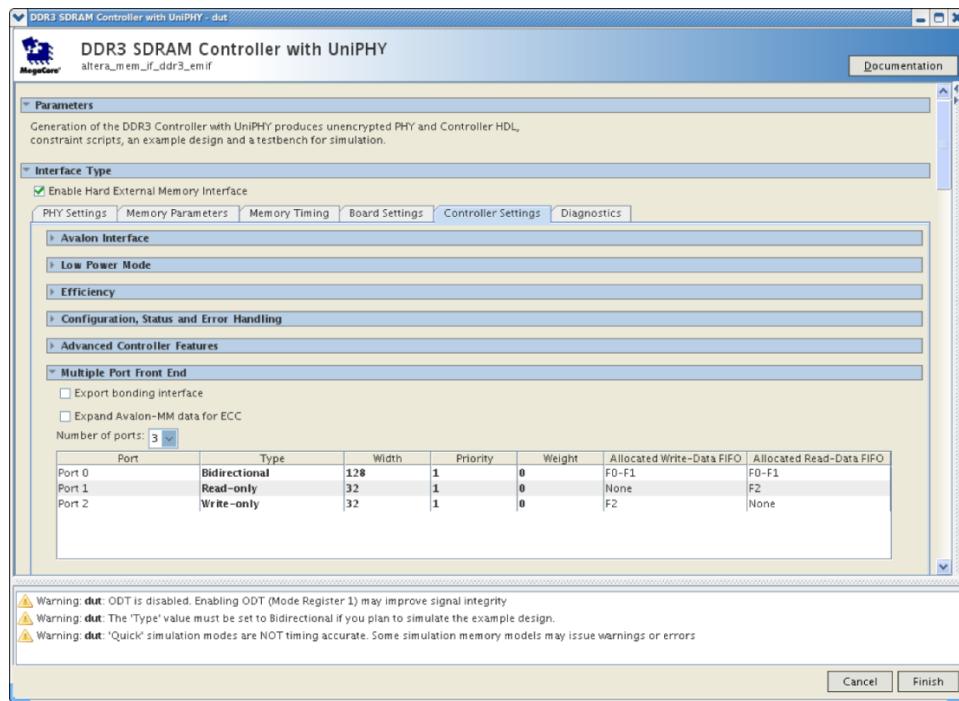
By default, the IP generates all clock signals regardless of the MPFE settings, but all unused ports and FIFO buffers are connected to ground.

The command ports can be used only in unidirectional configurations, with either 4 write and 2 read, 3 write and 3 read, or 2 write and 4 read scenarios. For bidirectional ports, the number of clocks is reduced from 6 to a maximum of 4.

For the scenario depicted in the following figure:

- command port 0 is associated with read and write data FIFO 0 and 1
- command port 1 is associated with read data FIFO 2
- command port 2 is associated with write data FIFO 2

**Figure 144. Sample MPFE Configuration**



Therefore, if port 0 (avl\_0) is clocked by a 100 MHz clock signal, mp\_cmd\_clk\_0, mp\_rfifo\_clk\_0, mp\_rfifo\_clk\_1, mp\_wfifo\_clk\_0, and mp\_wfifo\_clk\_1 must all be connected to the same 100 MHz clock, as illustrated below.



Figure 145. Sample Connection Mapping

| Use                                 | Connections        | Name                      | Description                       |
|-------------------------------------|--------------------|---------------------------|-----------------------------------|
| <input checked="" type="checkbox"/> |                    | <b>100_MHz</b>            | Clock Source                      |
|                                     | clk_in             | clk_in                    | Clock Input                       |
|                                     | clk_in_reset       | clk_in_reset              | Reset Input                       |
|                                     | clk                | clk                       | Clock Output                      |
|                                     | clk_reset          | clk_reset                 | Reset Output                      |
| <input checked="" type="checkbox"/> |                    | <b>125_MHz</b>            | Clock Source                      |
|                                     | clk_in             | clk_in                    | Clock Input                       |
|                                     | clk_in_reset       | clk_in_reset              | Reset Input                       |
|                                     | clk                | clk                       | Clock Output                      |
|                                     | clk_reset          | clk_reset                 | Reset Output                      |
| <input checked="" type="checkbox"/> |                    | <b>200_MHz</b>            | Clock Source                      |
|                                     | clk_in             | clk_in                    | Clock Input                       |
|                                     | clk_in_reset       | clk_in_reset              | Reset Input                       |
|                                     | clk                | clk                       | Clock Output                      |
|                                     | clk_reset          | clk_reset                 | Reset Output                      |
| <input checked="" type="checkbox"/> |                    | <b>mem_if_ddr3_emif_0</b> | DDR3 SDRAM Controller with UniPHY |
|                                     | pll_ref_clk        | pll_ref_clk               | Clock Input                       |
|                                     | global_reset       | global_reset              | Reset Input                       |
|                                     | soft_reset         | soft_reset                | Reset Input                       |
|                                     | afi_clk            | afi_clk                   | Clock Output                      |
|                                     | afi_half_clk       | afi_half_clk              | Clock Output                      |
|                                     | afi_reset          | afi_reset                 | Reset Output                      |
|                                     | memory             | memory                    | Conduit                           |
|                                     | →                  | avl_0                     | Avalon Memory Mapped Slave        |
|                                     | →                  | avl_1                     | Avalon Memory Mapped Slave        |
|                                     | →                  | avl_2                     | Avalon Memory Mapped Slave        |
|                                     | mp_cmd_clk_0       | mp_cmd_clk_0              | Clock Input                       |
|                                     | mp_cmd_reset_n_0   | mp_cmd_reset_n_0          | Reset Input                       |
|                                     | mp_cmd_clk_1       | mp_cmd_clk_1              | Clock Input                       |
|                                     | mp_cmd_reset_n_1   | mp_cmd_reset_n_1          | Reset Input                       |
|                                     | mp_cmd_clk_2       | mp_cmd_clk_2              | Clock Input                       |
|                                     | mp_cmd_reset_n_2   | mp_cmd_reset_n_2          | Reset Input                       |
|                                     | mp_rfifo_clk_0     | mp_rfifo_clk_0            | Clock Input                       |
|                                     | mp_rfifo_reset_n_0 | mp_rfifo_reset_n_0        | Reset Input                       |
|                                     | mp_wfifo_clk_0     | mp_wfifo_clk_0            | Clock Input                       |
|                                     | mp_wfifo_reset_n_0 | mp_wfifo_reset_n_0        | Reset Input                       |
|                                     | mp_rfifo_clk_1     | mp_rfifo_clk_1            | Clock Input                       |
|                                     | mp_rfifo_reset_n_1 | mp_rfifo_reset_n_1        | Reset Input                       |
|                                     | mp_wfifo_clk_1     | mp_wfifo_clk_1            | Clock Input                       |
|                                     | mp_rfifo_reset_n_2 | mp_rfifo_reset_n_2        | Reset Input                       |
|                                     | mp_wfifo_clk_2     | mp_wfifo_clk_2            | Clock Input                       |
|                                     | mp_rfifo_reset_n_2 | mp_rfifo_reset_n_2        | Reset Input                       |
|                                     | status             | status                    | Conduit                           |
|                                     | oct                | oct                       | Conduit                           |

## 6.4. Hard Memory Controller

The hard memory controller initializes, refreshes, manages, and communicates with the external memory device.

**Note:** The hard memory controller is functionally similar to the High Performance Controller II (HPC II). For information on signals, refer to the *Functional Description—HPC II Controller* chapter.



### Related Information

Functional Description—HPC II Controller on page 337

#### 6.4.1. Clocking

The ports on the MPFE can be clocked at different frequencies, and synchronization is maintained by cross-domain clocking logic in the MPFE.

Command ports can connect to different clocks, but the data ports associated with a given command port must be attached to the same clock as that command port. For example, a bidirectional command port that performs a 64-bit read/write function has its read port and write port connected to the same clock as the command port. Note that these clocks are separate from the EMIF core generated clocks.

#### 6.4.2. Reset

The ports of the MPFE must connect to the same reset signal.

When the reset signal is asserted, it resets the command and data FIFO buffer in the MPFE without resetting the hard memory controller.

*Note:*

The `global_reset_n` and `soft_reset_n` signals are asynchronous.

For easiest management of reset signals, Intel recommends the following sequence at power-up:

1. Initially `global_reset_n`, `soft_reset_n`, and the MPFE reset signals are all asserted.
2. `global_reset_n` is deasserted.
3. Wait for `pll_locked` to transition high.
4. `soft_reset_n` is deasserted.
5. (Optional) If you encounter difficulties, wait for the controller signal `local_cal_success` to go high, indicating that the external memory interface has successfully completed calibration, before deasserting the MPFE FIFO reset signals. This will ensure that read/write activity cannot occur until the interface is successfully calibrated.

#### 6.4.3. DRAM Interface

The DRAM interface is 40 bits wide, and can accommodate 8-bit, 16-bit, 16-bit plus ECC, 32-bit, or 32-bit plus ECC configurations. Any unused I/Os in the DRAM interface can be reused as user I/Os.

The DRAM interface supports DDR2 and DDR3 memory protocols, and LPDDR2 for Cyclone V only. Fast and medium speed grade devices are supported to 533 MHz for Arria V and 400 MHz for Cyclone V.

#### 6.4.4. ECC

The hard controller supports both error-correcting code (ECC) calculated by the controller and by the user. Controller ECC code employs standard Hamming logic which can detect and correct single-bit errors and detect double-bit errors. The



controller ECC is available for 16-bit and 32-bit widths, each requiring an additional 8 bits of memory, resulting in an actual memory width of 24-bits and 40-bits, respectively.

In user ECC mode, all bits are treated as data bits, and are written to and read from memory. User ECC can implement nonstandard memory widths such as 24-bit or 40-bit, where ECC is not required.

### Controller ECC

Controller ECC provides the following features:

**Byte Writes**—The memory controller performs a read/modify/write operation to keep ECC valid when a subset of the bits of a word is being written. If an entire word is being written (but less than a full burst) and the DM pins are connected, no read is necessary and only that word is updated. If controller ECC is disabled, byte-writes have no performance impact.

**ECC Write Backs**—When a read operation detects a correctable error, the memory location is scheduled for a read/modify/write operation to correct the single-bit error.

**User ECC**—User ECC is 24-bits or 40-bits wide; with user ECC, the controller performs no ECC checking. The controller employs memory word addressing with byte enables, and can handle arbitrary memory widths. User ECC does not disable byte writes; hence, you must ensure that any byte writes do not result in corrupted ECC.

### 6.4.5. Bonding of Memory Controllers

Bonding is a feature that allows data to be split between two memory controllers, providing the ability to service bandwidth streams similar to a single 64-bit controller. Bonding works by dividing data buses in proportion to the memory widths, and always sending a transaction to both controllers. When signals are returned, bonding ensures that both sets of signals are returned identically.

Bonding can be applied to asymmetric controllers, and allows controllers to have different memory clocks. Bonding does not attempt to synchronize the controllers. Bonding supports only one port. The Avalon port width can be varied from 64-bit to 256-bit; 32-bit port width is not supported.

The following signals require bonding circuitry:

**Read data return**—This bonding allows read data from the two controllers to return with effectively one ready signal to the bus master that initiated the bus transaction.

**Write ready**—For Avalon-MM, this is effectively bonding on the waitrequest signal.

**Write acknowledge**—Synchronization on returning the write completed signal.

For each of the above implementations, data is returned in order, hence the circuitry must match up for each valid cycle.

Bonded FIFO buffers must have identical FIFO numbers; that is, read FIFO 1 on controller 1 must be paired with Read FIFO 1 on controller 2.



## Data Return Bonding

Long loop times can lead to communications problems when using bonded controllers. The following effects are possible when using bonded controllers:

- If one memory controller completes its transaction and receives new data before the other controller, then the second controller can send data as soon as it arrives, and before the first controller acknowledges that the second controller has data.
- If the first controller has a single word in its FIFO buffer and the second controller receives single-word transactions, the second controller must determine whether the second word is a valid signal or not.

To accommodate the above effects, the hard controller maintains two counters for each bonded pair of FIFO buffers and implements logic that monitors those counters to ensure that the bonded controllers receive the same data on the same cycle, and that they send the data out on the same cycle.

## FIFO Ready

FIFO ready bonding is used for write command and write data buses. The implementation is similar to the data return bonding.

## Bonding Latency Impact

Bonding has no latency impact on ports that are not bonded.

## Bonding Controller Usage

Arria V and Cyclone V devices employ three shared bonding controllers to manage the read data return bonding, write acknowledge bonding, and command/write data ready bonding.

The three bonding controllers require three pairs of bonding I/Os, each based on a six port count; this means that a bonded hard memory controller requires 21 input signals and 21 output signals for its connection to the fabric, and another 21 input signals and 21 output signals to the paired hard memory controller.

*Note:* The hard processor system (HPS) hard memory controller cannot be bonded with another hard memory controller on the FPGA portion of the device.

## Bonding Configurations and Parameter Requirements

Intel has verified hard memory controller bonding between two interfaces with the following configuration:

- Same clock source
- Same memory clock frequency
- Same memory parameters and timings (except interface width)
- Same controller settings.
- Same port width in MPFE settings

Bonding supports only one port. The Avalon port width can be varied from 64-bits to 256-bits; a 32-bit port width is not supported.



## 6.5. Hard PHY

A physical layer interface (PHY) is embedded in the periphery of the Arria V device, and can run at the same high speed as the hard controller and hard sequencer. This hard PHY is located next to the hard controller. Differing device configurations have different numbers and sizes of hard controller and hard PHY pairs.

The hard PHY implements logic that connects the hard controller to the I/O ports. Because the hard controller and AFI interface support high frequencies, a portion of the sequencer is implemented as hard logic. The Nios II processor, the instruction/data RAM, and the Avalon fabric of the sequencer are implemented as core soft logic. The read/write manger and PHY manager components of the sequencer, which must operate at full rate, are implemented as hard logic in the hard PHY.

### 6.5.1. Interconnections

The hard PHY resides on the device between the hard controller and the I/O register blocks. The hard PHY is instantiated or bypassed entirely, depending on the parameterization that you specify.

The hard PHY connects to the hard memory controller and the core, enabling the use of either the hard memory controller or a software-based controller. (You can have the hard controller and hard PHY, or the soft controller and soft PHY; however, the combination of soft controller with hard PHY is not supported.) The hard PHY also connects to the I/O register blocks and the DQS logic. The path between the hard PHY and the I/O register blocks can be bypassed, but not reconfigured—in other words, if you use the hard PHY datapath, the pins to which it connects are predefined and specified by the device pin table.

### 6.5.2. Clock Domains

The hard PHY contains circuitry that uses the following clock domains:

**AFI clock domain (pll\_afi\_clk)** —The main full-rate clock signal that synchronizes most of the circuit logic.

**Avalon clock domain (pll\_avl\_clk)** —Synchronizes data on the internal Avalon bus, namely the Read/Write Manager, PHY Manager, and Data Manager data. The data is then transferred to the AFI clock domain. To ensure reliable data transfer between clock domains, the Avalon clock period must be an integer multiple of the AFI clock period, and the phases of the two clocks must be aligned.

**Address and Command clock domain (pll\_addr\_cmd\_clk)** —Synchronizes the global asynchronous reset signal, used by the I/Os in this clock domain.

### 6.5.3. Hard Sequencer

The sequencer initializes the memory device and calibrates the I/Os, with the objective of maximizing timing margins and achieving the highest possible performance.



When the hard memory controller is in use, a portion of the sequencer must run at full rate; for this reason, the Read/Write Manager, PHY Manager, and Data Manager are implemented as hard components within the hard PHY. The hard sequencer communicates with the soft-logic sequencer components (including the Nios II processor) via an Avalon bus.

#### 6.5.4. MPFE Setup Guidelines

The following instructions provide information on configuring the multi-port front end of the hard memory interface.

1. To enable the hard memory interface, turn on **Enable Hard External Memory Interface** in the **Interface Type** tab in the parameter editor.
2. To export bonding interface ports to the top level, turn on **Export bonding interface** in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor.

*Note:* The system exports three bonding-in ports and three bonding-out ports. You must generate two controllers and connect the bonding ports manually.

3. To expand the interface data width from a maximum of 32 bits to a maximum of 40 bits, turn on **Enable Avalon-MM data for ECC** in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor.

*Note:* The controller does not perform ECC checking when this option is turned on.

4. Select the required **Number of ports** for the multi-port front end in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor.

*Note:* The maximum number of ports is 6, depending on the port type and width. The maximum port width is 256 bits, which is the maximum data width of the read data FIFO and write data FIFO buffers.

5. The table in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor lists the ports that are created. The columns in the table describe each port, as follows:

- **Port:** Indicates the port number.
- **Type:** Indicates whether the port is read only, write only, or bidirectional.
- **Width:** To achieve optimum MPFE throughput, Intel recommends setting the MPFE data port width according to the following calculation:

$$2 \times (\text{frequency ratio of HMC to user logic}) \times (\text{interface data width})$$

For example, if the frequency of your user logic is one-half the frequency of the hard memory controller, you should set the port width to be 4x the interface data width. If the frequency ratio of the hard memory controller to user logic is a fractional value, you should use a larger value; for example, if the ratio is 1.5, you can use 2.



- **Priority:** The priority setting specifies the priority of the slave port, with higher values representing higher priority. The slave port with highest priority is served first.
- **Weight:** The weight setting has a range of values of 0–31, and specifies the relative priority of a slave port, with higher weight representing higher priority. The weight value can determine relative bandwidth allocations for slave ports with the same priority values. For example, if two ports have the same priority value, and weight values of 4 and 6, respectively, the port with a weight of 4 will receive 40% of the bus bandwidth, while the port with a weight of 6 will receive 60% of the bus bandwidth—assuming 100% total available bus bandwidth.

### 6.5.5. Soft Memory Interface to Hard Memory Interface Migration Guidelines

The following instructions provide information on mapping your soft memory interface to a hard memory interface.

#### Pin Connections

1. The hard and soft memory controllers have compatible pinouts. Assign interface pins to the hard memory interface according to the pin table.
2. Ensure that your soft memory interface pins can fit into the hard memory interface. The hard memory interface can support a maximum of a 40-bit interface with user ECC, or a maximum of 80-bits with same-side bonding. The soft memory interface does not support bonding.
3. Follow the recommended board layout guidelines for the hard memory interface.

#### Software Interface Preparation

Observe the following points in preparing your soft memory interface for migration to a hard memory interface:

- You cannot use the hard PHY without also using the hard memory controller.
- The hard memory interface supports only full-rate controller mode.
- Ensure that the MPFE data port width is set according to the soft memory interface half-rate mode Avalon data width.
- The hard memory interface uses a different Avalon port signal naming convention than the software memory interface. Ensure that you change the `avl_<signal_name>` signals in the soft memory interface to `.avl_<signal_name>_0` signals in the hard memory interface.
- The hard memory controller MPFE includes an additional three clocks and three reset ports (CMD port, RFIFO port, and WFIFO port) that do not exist with the soft memory controller. You should connect the user logic clock signal to the MPFE clock port, and the user logic reset signal to the MPFE reset port.
- In the soft memory interface, the half-rate `afi_clk` is a user logic clock. In the hard memory interface, `afi_clk` is a full-rate clock, because the core fabric might not be able to achieve full-rate speed. When you migrate your soft memory interface to a hard memory interface, you need to supply an additional slower rate clock. The maximum clock rate supported by core logic is one-half of the maximum interface frequency.

#### Latency



Overall, you should expect to see slightly more latency when using the hard memory controller and multi-port front end, than when using the soft memory controller.

The hard memory controller typically exhibits lower latency than the soft memory controller; however, the multi-port front end does introduce additional latency cycles due to FIFO buffer stages used for synchronization. The MPFE cannot be bypassed, even if only one port is needed.

### 6.5.6. Bonding Interface Guidelines

Bonding allows a single data stream to be split between two memory controllers, providing the ability to expand the interface data width similar to a single 64-bit controller. This section provides some guidelines for setting up the bonding interface.

1. Bonding interface ports are exported to the top level in your design. You should connect each `bonding_in*` port in one hard memory controller to the corresponding `bonding_out_*` port in the other hard memory controller, and vice versa.
2. You should modify the Avalon signal connections to drive the bonding interface with a single user logic/master, as follows:
  - a. AND both `avl_ready` signals from both hard memory controllers before the signals enter the user logic.
  - b. AND both `avl_rdata_valid` signals from both hard memory controllers before the signals enter the user logic. (The `avl_rdata_valid` signals should be identical for both hard memory controllers.)
  - c. Branch the following signals from the user logic to both hard memory controllers:
    - `avl_burstbegin`
    - `avl_addr`
    - `avl_read_req`
    - `avl_write_req`
    - `avl_size`
  - d. Split the following signals according to each multi-port front end data port width:
    - `avl_rdata`
    - `avl_wdata`
    - `avl_be`

### 6.6. Document Revision History

| Date          | Version    | Changes              |
|---------------|------------|----------------------|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release. |
| May 2016      | 2016.05.02 | Maintenance release. |
| November 2015 | 2015.11.02 | Maintenance release. |

*continued...*



| Date          | Version     | Changes  |
|---------------|-------------|--|
| May 2015      | 2015.05.04  | Maintenance release.   |
| December 2014 | 21014.12.15 | Maintenance release.   |
| August 2014   | 2014.08.15  | <ul style="list-style-type: none"> <li>Updated descriptions of <code>mp_cmd_reset_n_#_reset_n</code>, <code>mp_rfifo_reset_n_#_reset_n</code>, and <code>mp_wfifo_reset_n_#_reset_n</code> in the <i>MPFE Signals</i> table.</li> <li>Added <i>Reset</i> section to <i>Hard Memory Controller</i> section.</li> </ul>    |
| December 2013 | 2013.12.16  | <ul style="list-style-type: none"> <li>Added footnote about command FIFOs to MPFE Signals table.</li> <li>Added information about FIFO depth for the MPFE.</li> <li>Added information about hard memory controller bonding.</li> <li>Reworded protocol-support information for Arria V and Cyclone V devices.</li> </ul> |
| November 2012 | 2.1         | <ul style="list-style-type: none"> <li>Added <i>Hard Memory Interface Implementation Guidelines</i>.</li> <li>Moved content of EMI-Related HPS Features in SoC Devices section to chapter 4, <i>Functional Description—HPS Memory Controller</i>.</li> </ul>   |
| June 2012     | 2.0         | <ul style="list-style-type: none"> <li>Added EMI-Related HPS Features in SoC Devices.</li> <li>Added LPDDR2 support.</li> <li>Added Feedback icon.</li> </ul>  |
| November 2011 | 1.0         | Initial release.   |

## 7. Functional Description—HPS Memory Controller

---

The hard processor system (HPS) SDRAM controller subsystem provides efficient access to external SDRAM for the ARM® Cortex\*-A9 microprocessor unit (MPU) subsystem, the level 3 (L3) interconnect, and the FPGA fabric.

**Note:** This chapter applies to the HPS architecture of Arria V and Cyclone V memory controllers only.

The SDRAM controller provides an interface between the FPGA fabric and HPS. The interface accepts Advanced Microcontroller Bus Architecture (AMBA®) Advanced eXtensible Interface (AXI™) and Avalon® Memory-Mapped (Avalon-MM) transactions, converts those commands to the correct commands for the SDRAM, and manages the details of the SDRAM access.

### 7.1. Features of the SDRAM Controller Subsystem

The SDRAM controller subsystem offers programming flexibility, port and bus configurability, error correction, and power management for external memories up to 4 GB.

- Support for double data rate 2 (DDR2), DDR3, and low-power DDR2 (LPDDR2) SDRAM
- Flexible row and column addressing with the ability to support up to 4 GB of memory in various interface configurations
- Optional 8-bit integrated error correction code (ECC) for 16- and 32-bit data widths<sup>(3)</sup>
- User-configurable memory width of 8, 16, 16+ECC, 32, 32+ECC
- User-configurable timing parameters
- Two chip selects (DDR2 and DDR3)
- Command reordering (look-ahead bank management)
- Data reordering (out of order transactions)
- User-controllable bank policy on a per port basis for either closed page or conditional open page accesses
- User-configurable priority support with both absolute and weighted round-robin scheduling
- Flexible FPGA fabric interface with up to 6 ports that can be combined for a data width up to 256 bits using Avalon-MM and AXI interfaces
- Power management supporting self refresh, partial array self refresh (PASR), power down, and LPDDR2 deep power down

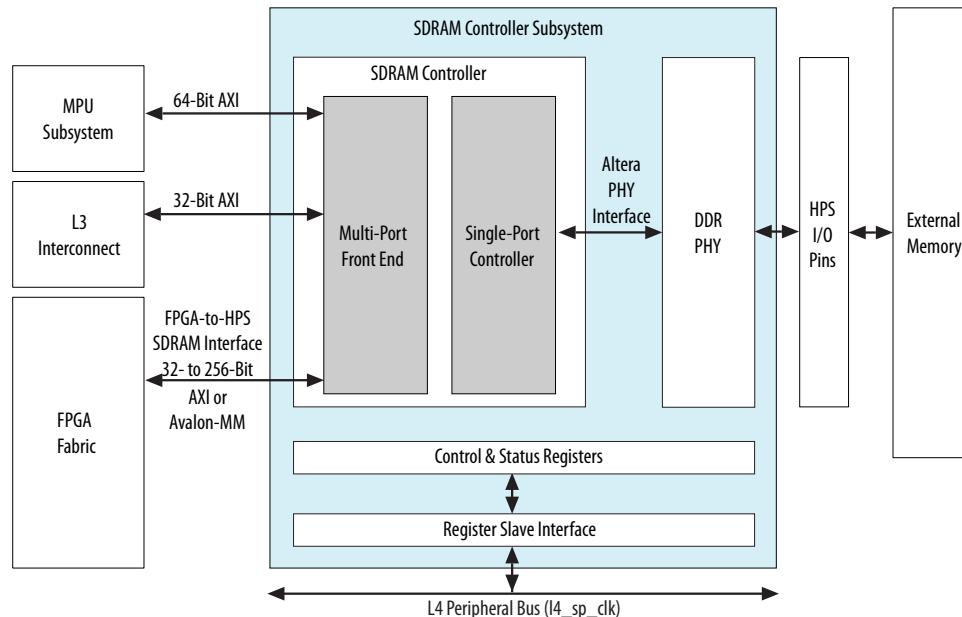
---

<sup>(3)</sup> The level of ECC support is package dependent.

## 7.2. SDRAM Controller Subsystem Block Diagram

The SDRAM controller subsystem connects to the MPU subsystem, the L3 interconnect, and the FPGA fabric. The memory interface consists of the SDRAM controller, the physical layer (PHY), control and status registers (CSRs), and their associated interfaces.

**Figure 146. SDRAM Controller Subsystem High-Level Block Diagram**



### SDRAM Controller

The SDRAM controller provides high performance data access and run-time programmability. The controller reorders data to reduce row conflicts and bus turn-around time by grouping read and write transactions together, allowing for efficient traffic patterns and reduced latency.

The SDRAM controller consists of a multiport front end (MPFE) and a single-port controller. The MPFE provides multiple independent interfaces to the single-port controller. The single-port controller communicates with and manages each external memory device.

The MPFE FPGA-to-HPS SDRAM interface port has an asynchronous FIFO buffer followed by a synchronous FIFO buffer. Both the asynchronous and synchronous FIFO buffers have a read and write data FIFO depth of 8, and a command FIFO depth of 4. The MPU subsystem 64-bit AXI port and L3 interconnect 32-bit AXI port have asynchronous FIFO buffers with read and write data FIFO depth of 8, and command FIFO depth of 4.



## DDR PHY

The DDR PHY provides a physical layer interface for read and write memory operations between the memory controller and memory devices. The DDR PHY has dataflow components, control components, and calibration logic that handle the calibration for the SDRAM interface timing.

### Related Information

[Memory Controller Architecture](#) on page 272

## 7.3. SDRAM Controller Memory Options

Bank selects, and row and column address lines can be configured to work with SDRAMs of various technology and density combinations.

**Table 64. SDRAM Controller Interface Memory Options**

| Memory Type<br><sup>(4)</sup> | Mbits                           | Column Address Bit Width | Bank Select Bit Width | Row Address Bit Width | Page Size | MBytes |
|-------------------------------|---------------------------------|--------------------------|-----------------------|-----------------------|-----------|--------|
| DDR2                          | 256                             | 10                       | 2                     | 13                    | 1024      | 32     |
|                               | 512                             | 10                       | 2                     | 14                    | 1024      | 64     |
|                               | 1024 (1 Gb)                     | 10                       | 3                     | 14                    | 1024      | 128    |
|                               | 2048 (2 Gb)                     | 10                       | 3                     | 15                    | 1024      | 256    |
|                               | 4096 (4 Gb)                     | 10                       | 3                     | 16                    | 1024      | 512    |
| DDR3                          | 512                             | 10                       | 3                     | 13                    | 1024      | 64     |
|                               | 1024 (1 Gb)                     | 10                       | 3                     | 14                    | 1024      | 128    |
|                               | 2048 (2 Gb)                     | 10                       | 3                     | 15                    | 1024      | 256    |
|                               | 4096 (4 Gb)                     | 10                       | 3                     | 16                    | 1024      | 512    |
| LPDDR2                        | 64                              | 9                        | 2                     | 12                    | 512       | 8      |
|                               | 128                             | 10                       | 2                     | 12                    | 1024      | 16     |
|                               | 256                             | 10                       | 2                     | 13                    | 1024      | 32     |
|                               | 512                             | 11                       | 2                     | 13                    | 2048      | 64     |
|                               | 1024 (1 Gb) - S2 <sup>(5)</sup> | 11                       | 2                     | 14                    | 2048      | 128    |
|                               | 1024 (1 Gb) - S4 <sup>(6)</sup> | 11                       | 3                     | 13                    | 2048      | 128    |
|                               | 2048 (2 Gb) - S2 <sup>(5)</sup> | 11                       | 2                     | 15                    | 2048      | 256    |
|                               | 2048                            | 11                       | 3                     | 14                    | 2048      | 256    |

*continued...*

(4) For all memory types shown in this table, the DQ width is 8.

(5) S2 signifies a 2n prefetch size

(6) S4 signifies a 4n prefetch size



| Memory Type<br>( <sup>4</sup> ) | Mbits                       | Column Address Bit Width | Bank Select Bit Width | Row Address Bit Width | Page Size | MBytes |
|---------------------------------|-----------------------------|--------------------------|-----------------------|-----------------------|-----------|--------|
|                                 | (2 Gb) -S4 ( <sup>6</sup> ) |                          |                       |                       |           |        |
|                                 | 4096 (4 Gb)                 | 12                       | 3                     | 14                    | 4096      | 512    |

## 7.4. SDRAM Controller Subsystem Interfaces

### 7.4.1. MPU Subsystem Interface

The SDRAM controller connects to the MPU subsystem with a dedicated 64-bit AXI interface, operating on the `mpu_12_ram_clk` clock domain.

### 7.4.2. L3 Interconnect Interface

The SDRAM controller interfaces to the L3 interconnect with a dedicated 32-bit AXI interface, operating on the `l3_main_clk` clock domain.

### 7.4.3. CSR Interface

The CSR interface connects to the level 4 (L4) bus and operates on the `l4_sp_clk` clock domain. The MPU subsystem uses the CSR interface to configure the controller and PHY, for example setting the memory timing parameter values or placing the memory in a low power state. The CSR interface also provides access to the status registers in the controller and PHY.

### 7.4.4. FPGA-to-HPS SDRAM Interface

The FPGA-to-HPS SDRAM interface provides masters implemented in the FPGA fabric access to the SDRAM controller subsystem in the HPS. The interface has three port types that are used to construct the following AXI or Avalon-MM interfaces:

- Command ports—issue read and/or write commands, and for receive write acknowledge responses
- 64-bit read data ports—receive data returned from a memory read
- 64-bit write data ports—transmit write data

The FPGA-to-HPS SDRAM interface supports six command ports, allowing up to six Avalon-MM interfaces or three AXI interfaces. Each command port can be used to implement either a read or write command port for AXI, or be used as part of an Avalon-MM interface. The AXI and Avalon-MM interfaces can be configured to support 32-, 64-, 128-, and 256-bit data.

---

<sup>(4)</sup> For all memory types shown in this table, the DQ width is 8.

**Table 65. FPGA-to-HPS SDRAM Controller Port Types**

| Port Type         | Available Number of Ports |
|-------------------|---------------------------|
| Command           | 6                         |
| 64-bit read data  | 4                         |
| 64-bit write data | 4                         |

The FPGA-to-HPS SDRAM controller interface can be configured with the following characteristics:

- Avalon-MM interfaces and AXI interfaces can be mixed and matched as required by the fabric logic, within the bounds of the number of ports provided to the fabric.
- Because the AXI protocol allows simultaneous read and write commands to be issued, two SDRAM control ports are required to form an AXI interface.
- Because the data ports are natively 64-bit, they must be combined if wider data paths are required for the interface.
- Each Avalon-MM or AXI interface of the FPGA-to-HPS SDRAM interface operates on an independent clock domain.
- The FPGA-to-HPS SDRAM interfaces are configured during FPGA configuration.

The following table shows the number of ports needed to configure different bus protocols, based on type and data width.

**Table 66. FPGA-to-HPS SDRAM Port Utilization**

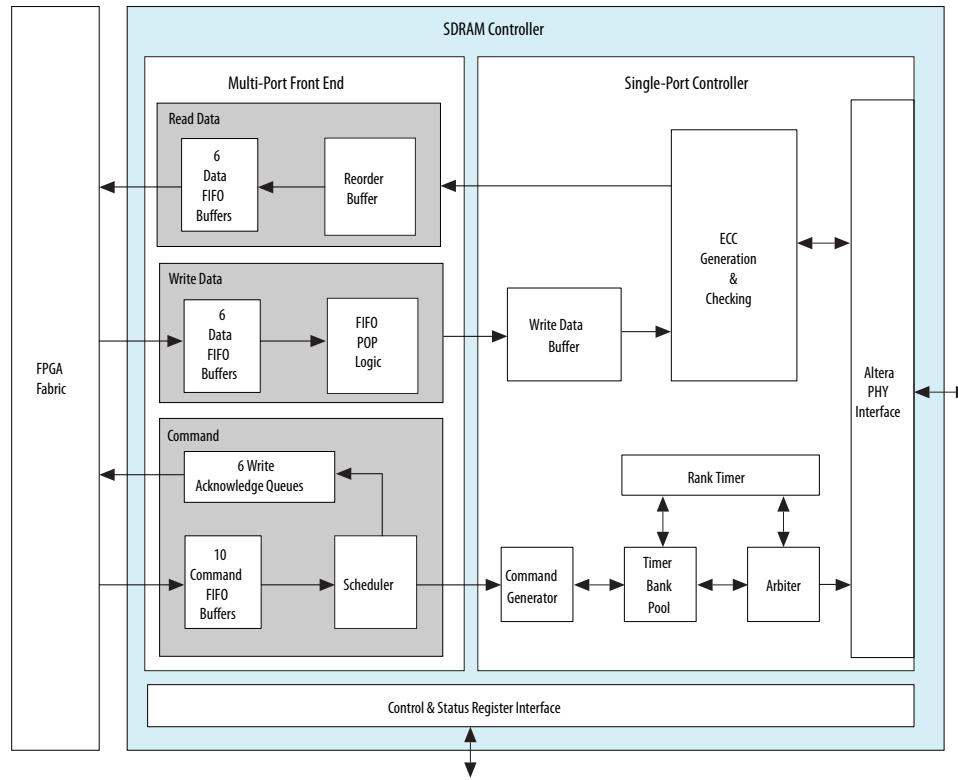
| Bus Protocol                       | Command Ports | Read Data Ports | Write Data Ports |
|------------------------------------|---------------|-----------------|------------------|
| 32- or 64-bit AXI                  | 2             | 1               | 1                |
| 128-bit AXI                        | 2             | 2               | 2                |
| 256-bit AXI                        | 2             | 4               | 4                |
| 32- or 64-bit Avalon-MM            | 1             | 1               | 1                |
| 128-bit Avalon-MM                  | 1             | 2               | 2                |
| 256-bit Avalon-MM                  | 1             | 4               | 4                |
| 32- or 64-bit Avalon-MM write-only | 1             | 0               | 1                |
| 128-bit Avalon-MM write-only       | 1             | 0               | 2                |
| 256-bit Avalon-MM write-only       | 1             | 0               | 4                |
| 32- or 64-bit Avalon-MM read-only  | 1             | 1               | 0                |
| 128-bit Avalon-MM read-only        | 1             | 2               | 0                |
| 256-bit Avalon-MM read-only        | 1             | 4               | 0                |

## 7.5. Memory Controller Architecture

The SDRAM controller consists of an MPFE, a single-port controller, and an interface to the CSRs.



**Figure 147. SDRAM Controller Block Diagram**



### 7.5.1. Multi-Port Front End

The Multi-Port Front End (MPFE) is responsible for scheduling pending transactions from the configured interfaces and sending the scheduled memory transactions to the single-port controller. The MPFE handles all functions related to individual ports.

The MPFE consists of three primary sub-blocks.

#### Command Block

The command block accepts read and write transactions from the FPGA fabric and the HPS. When the command FIFO buffer is full, the command block applies backpressure by deasserting the ready signal. For each pending transaction, the command block calculates the next SDRAM burst needed to progress on that transaction. The command block schedules pending SDRAM burst commands based on the user-supplied configuration, available write data, and unallocated read data space.

#### Write Data Block

The write data block transmits data to the single-port controller. The write data block maintains write data FIFO buffers and clock boundary crossing for the write data. The write data block informs the command block of the amount of pending write data for each transaction so that the command block can calculate eligibility for the next SDRAM write burst.

### Read Data Block

The read data block receives data from the single-port controller. Depending on the port state, the read data block either buffers the data in its internal buffer or passes the data straight to the clock boundary crossing FIFO buffer. The read data block reorders out-of-order data for Avalon-MM ports.

In order to prevent the read FIFO buffer from overflowing, the read data block informs the command block of the available buffer area so the command block can pace read transaction dispatch.

## 7.5.2. Single-Port Controller

The single-port logic is responsible for following actions:

- Queuing the pending SDRAM bursts
- Choosing the most efficient burst to send next
- Keeping the SDRAM pipeline full
- Ensuring all SDRAM timing parameters are met

Transactions passed to the single-port logic for a single page in SDRAM are guaranteed to be executed in order, but transactions can be reordered between pages. Each SDRAM burst read or write is converted to the appropriate Altera PHY interface (AFI) command to open a bank on the correct row for the transaction (if required), execute the read or write command, and precharge the bank (if required).

The single-port logic implements command reordering (looking ahead at the command sequence to see which banks can be put into the correct state to allow a read or write command to be executed) and data reordering (allowing data transactions to be dispatched even if the data transactions are executed in an order different than they were received from the multi-port logic).

The single-port controller consists of eight sub-modules.

### 7.5.2.1. Command Generator

The command generator accepts commands from the MPFE and from the internal ECC logic, and provides those commands to the timer bank pool.

#### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.

### 7.5.2.2. Timer Bank Pool

The timer bank pool is a parallel queue that operates with the arbiter to enable data reordering. The timer bank pool tracks incoming requests, ensures that all timing requirements are met, and, on receiving write-data-ready notifications from the write data buffer, passes the requests to the arbiter

#### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.



### 7.5.2.3. Arbiter

The arbiter determines the order in which requests are passed to the memory device. When the arbiter receives a single request, that request is passed immediately. When multiple requests are received, the arbiter uses arbitration rules to determine the order to pass requests to the memory device.

#### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.

### 7.5.2.4. Rank Timer

The rank timer performs the following functions:

- Maintains rank-specific timing information
- Ensures that only four activates occur within a specified timing window
- Manages the read-to-write and write-to-read bus turnaround time
- Manages the time-to-activate delay between different banks

#### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.

### 7.5.2.5. Write Data Buffer

The write data buffer receives write data from the MPFE and passes the data to the PHY, on approval of the write request.

#### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.

### 7.5.2.6. ECC Block

The ECC block consists of an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC block can correct single-bit errors and detect double-bit errors resulting from noise or other impairments during data transmission.

**Note:** The level of ECC support is package dependent.

#### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.

### 7.5.2.7. AFI Interface

The AFI interface provides communication between the controller and the PHY.



### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.

#### 7.5.2.8. CSR Interface

The CSR interface is accessible from the L4 bus. The interface allows code in the HPS MPU or soft IP cores in the FPGA fabric to configure and monitor the SDRAM controller.

### Related Information

[Memory Controller Architecture](#) on page 272

For more information, refer to the SDRAM Controller Block diagram.

## 7.6. Functional Description of the SDRAM Controller Subsystem

### 7.6.1. MPFE Operation Ordering

Operation ordering is defined and enforced within a port, but not between ports. All transactions received on a single port for overlapping addresses execute in order. Requests arriving at different ports have no guaranteed order of service, except when a first transaction has completed before the second arrives.

Avalon-MM does not support write acknowledgement. When a port is configured to support Avalon-MM, you should read from the location that was previously written to ensure that the write operation has completed. When a port is configured to support AXI, the master accessing the port can safely issue a read operation to the same address as a write operation as soon as the write has been acknowledged. To keep write latency low, writes are acknowledged as soon as the transaction order is guaranteed—meaning that any operations received on any port to the same address as the write operation are executed after the write operation.

To reduce read latency, the single-port logic can return read data out of order to the multi-port logic. The returned data is rearranged to its initial order on a per port basis by the multi-port logic and no traffic reordering occurs between individual ports.

#### Read Data Handling

The MPFE contains a read buffer shared by all ports. If a port is capable of receiving returned data then the read buffer is bypassed. If the size of a read transaction is smaller than twice the memory interface width, the buffer RAM cannot be bypassed. The lowest memory latency occurs when the port is ready to receive data and the full width of the interface is utilized.

### 7.6.2. MPFE Multi-Port Arbitration

The HPS SDRAM controller multi-port front end (MPFE) contains a programmable arbiter. The arbiter decides which MPFE port gains access to the single-port memory controller.

The SDRAM transaction size that is arbitrated is a burst of two beats. This burst size ensures that the arbiter does not favor one port over another when the incoming transaction size is a large burst.



The arbiter makes decisions based on two criteria: priority and weight. The priority is an absolute arbitration priority where the higher priority ports always win arbitration over the lower priority ports. Because multiple ports can be set to the same priority, the weight value refines the port choice by implementing a round-robin arbitration among ports set to the same priority. This programmable weight allows you to assign a higher arbitration value to a port in comparison to others such that the highest weighted port receives more transaction bandwidth than the lower weighted ports of the same priority.

Before arbitration is performed, the MPFE buffers are checked for any incoming transactions. The priority of each port that has buffered transactions is compared and the highest priority wins. If multiple ports are of the same highest priority value, the port weight is applied to determine which port wins. Because the arbiter only allows SDRAM-sized bursts into the single-port memory controller, large transactions may need to be serviced multiple times before the read or write command is fully accepted to the single-port memory controller. The MPFE supports dynamic tuning of the priority and weight settings for each port, with changes committed into the SDRAM controller at fixed intervals of time.

Arbitration settings are applied to each port of the MPFE. The memory controller supports a mix of Avalon-MM and AXI protocols. As defined in the "Port Mappings" section, the Avalon-MM ports consume a single command port while the AXI ports consume a pair of command ports to support simultaneous read and write transactions. In total, there are ten command ports for the MPFE to arbitrate. The following table illustrates the command port mapping within the HPS as well as the ports exposed to the FPGA fabric.

**Table 67. HPS SDRAM MPFE Command Port Mapping**

| Command Port | Allowed Functions  | Data Size              |
|--------------|--|------------------------|
| 0, 2, 4      | FPGA fabric AXI read command ports<br>FPGA fabric Avalon-MM read or write command ports  | 32-bit to 256-bit data |
| 1, 3, 5      | FPGA fabric AXI write command ports<br>FPGA fabric Avalon-MM read or write command ports |                        |
| 6            | L3 AXI read command port   | 32-bit data            |
| 7            | MPU AXI read command port  | 64-bit data            |
| 8            | L3 AXI write command port  | 32-bit data            |
| 9            | MPU AXI write command port   | 64-bit data            |

When the FPGA ports are configured for AXI, the command ports are always assigned in groups of two starting with even number ports 0, 2, or 4 assigned to the read command channel. For example, if you configure the first FPGA-to-SDRAM port as AXI and the second port as Avalon-MM, you can expect the following mapping:

- Command port 0 = AXI read
- Command port 1 = AXI write
- Command port 2 = Avalon-MM read and write

## Setting the MPFE Priority

The priority of each of the ten command ports is configured through the userpriority field of the mppriority register. This 30-bit register uses 3 bits per port to configure the priority. The lowest priority is 0x0 and the highest priority is 0x7. The bits are mapped in ascending order with bits [2:0] assigned to command port 0 and bits [29:27] assigned to command port 9.

## Setting the MPFE Static Weights

The static weight settings used in the round-robin command port priority scheme are programmed in a 128-bit field distributed among four 32-bit registers:

- mpweight\_0\_4
- mpweight\_1\_4
- mpweight\_2\_4
- mpweight\_3\_4

Each port is assigned a 5-bit value within the 128-bit field, such that port 0 is assigned to bits [4:0] of the mpweight\_0\_4 register, port 1 is assigned to bits [9:5] of the mpweight\_0\_4 register up to port 9, which is assigned to bits[49:45] contained in the mpweight\_1\_4 register. The valid weight range for each port is 0x0 to 0x1F, with larger static weights representing a larger arbitration share.

Bits[113:50] in the mpweight\_1\_4, mpweight\_2\_4 and mpweight\_3\_4 registers, hold the sum of weights for each priority. This 64-bit field is divided into eight fields of 8-bits, each representing the sum of static weights. Bits[113:50] are mapped in ascending order with bits [57:50] holding the sum of static weights for all ports with priority setting 0x0, and bits [113:106] holding the sum of static weights for all ports with priority setting 0x7.

## Example Using MPFE Priority and Weights

In this example, the following settings apply:

- FPGA MPFE port 0 is assigned to AXI read commands and port 1 is assigned to AXI write commands.
- FPGA MPFE port 2 is assigned to Avalon-MM read and write commands.
- L3 master ports (ports 6 and 8) and the MPU ports (port 7 and 9) are given the lowest priority but the MPU ports are configured with more arbitration static weight than the L3 master ports.
- The FPGA MPFE command ports are given the highest priority; however, AXI ports 0 and 1 are given a larger static weight because they carry the highest priority traffic in the entire system. Assigning a high priority and larger static weight ensures ports 0 and 1 will receive the highest quality-of-service (QoS).

The table below details the port weights and sum of weights.



**Table 68. SDRAM MPFE Port Priority, Weights and Sum of Weights**

| Priority | Weights |        |        |        |        |        |        |        |        |        | Sum of Weights |
|----------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|----------------|
| -        | Port 0  | Port 1 | Port 2 | Port 3 | Port 4 | Port 5 | Port 6 | Port 7 | Port 8 | Port 9 | -              |
| 1        | 10      | 10     | 5      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 25             |
| 0        | 0       | 0      | 0      | 0      | 0      | 0      | 1      | 4      | 1      | 4      | 10             |

If the FPGA-to-SDRAM ports are configured according to the table and if both ports are accessed concurrently, you can expect the AXI port to receive 80% of the total service. This value is determined by taking the sum of port 0 and 1 weights divided by the total weight for all ports of priority 1. The remaining 20% of bandwidth is allocated to the Avalon-MM port. With these port settings, any FPGA transaction buffered by the MPFE for either slave port blocks the MPU and L3 masters from having their buffered transactions serviced. To avoid transaction starvation, you should assign ports the same priority level and adjust the bandwidth allocated to each port by adjusting the static weights.

#### MPFE Weight Calculation

The MPFE applies a deficit round-robin arbitration scheme to determine which port is serviced. The larger the port weight, the more often it is serviced. Ports are serviced only when they have buffered transactions and are set to the highest priority of all the ports that also have buffered transactions. The arbiter determines which port to service by examining the running weight of all the same ports at the same priority level and the largest running weight is serviced.

Each time a port is drained of all transactions, its running weight is set to 0x80. Each time a port is serviced, the static weight is added and the sum of weights is subtracted from the running weight for that port. Each time a port is not serviced (same priority as another port but has a lower running weight), the static weight for the port is added to the running weight of the port for that particular priority level. The running weight additions and subtractions are only applied to one priority level, so any time ports of a different priority level are being serviced, the running weight of a lower priority port is not modified.



## MPFE Multi-Port Arbitration Considerations for Use

When using MPFE multi-port arbitration, the following considerations apply:

- To ensure that the dynamic weight value does not roll over when a port is serviced, the following equation should always be true:  
$$(\text{sum of weights} - \text{static weight}) < 128$$
If the running weight remains less than 128, arbitration for that port remains functional.
- The memory controller commits the priority and weight registers into the MPFE arbiter once every 10 SDRAM clock cycles. As a result, when the `mppriority` register and `mpweight_*_4` registers are configured at run time, the update interval can occur while the registers are still being written and ports can have different priority or weights than intended for a brief period. Because the `mppriority` and `mpweight_*_4` registers can be updated in a single 32-bit transaction, Intel recommends updating first to ensure that transactions that need to be serviced have the appropriate priority after the next update. Because the static weights are divided among four 32-bit registers
- In addition to the `mppriority` register and `mpweight_*_4` registers, the `remappriority` register adds another level of priority to the port scheduling. By programming bit N in the `priorityremap` field of the `remappriority` register, any port with an absolute priority N is sent to the front of the single port command queue and is serviced before any other transaction. Please refer to the `remappriority` register for more details.

The scheduler is work-conserving. Write operations can only be scheduled when enough data for the SDRAM burst has been received. Read operations can only be scheduled when sufficient internal memory is free and the port is not occupying too much of the read buffer.

### 7.6.3. MPFE SDRAM Burst Scheduling

SDRAM burst scheduling recognizes addresses that access the same row/bank combination, known as open page accesses. Operations to a page are served in the order in which they are received by the single-port controller. Selection of SDRAM operations is a two-stage process. First, each pending transaction must wait for its timers to be eligible for execution. Next, the transaction arbitrates against other transactions that are also eligible for execution.

The following rules govern transaction arbitration:

- High-priority operations take precedence over lower-priority operations
- If multiple operations are in arbitration, read operations have precedence over write operations
- If multiple operations still exist, the oldest is served first

A high-priority transaction in the SDRAM burst scheduler wins arbitration for that bank immediately if the bank is idle and the high-priority transaction's chip select, row, or column fields of the address do not match an address already in the single-port controller. If the bank is not idle, other operations to that bank yield until the high-priority operation is finished. If the chip select, row, and column fields match an earlier transaction, the high-priority transaction yields until the earlier transaction is completed.



### Clocking

The FPGA fabric ports of the MPFE can be clocked at different frequencies. Synchronization is maintained by clock-domain crossing logic in the MPFE. Command ports can operate on different clock domains, but the data ports associated with a given command port must be attached to the same clock as that command port.

**Note:** A command port paired with a read and write port to form an Avalon-MM interface must operate at the same clock frequency as the data ports associated with it.

#### 7.6.4. Single-Port Controller Operation

The single-port controller increases the performance of memory transactions through command and data re-ordering, enforcing bank policies, combining write operations and allowing burst transfers. Correction of single-bit errors and detection of double-bit errors is handled in the ECC module of the single-port Controller.

##### SDRAM Interface

The SDRAM interface is up to 40 bits wide and can accommodate 8-bit, 16-bit, 16-bit plus ECC, 32-bit, or 32-bit plus ECC configurations, depending on the device package. The SDRAM interface supports LPDDR2, DDR2, and DDR3 memory protocols.

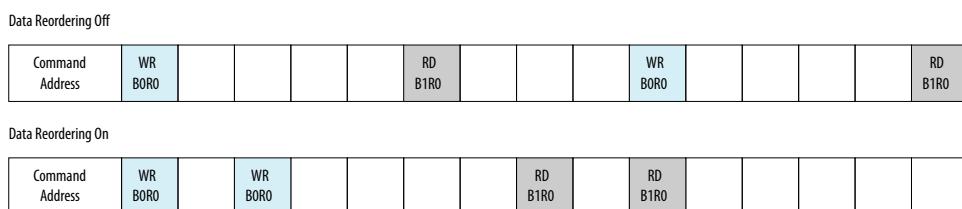
##### 7.6.4.1. Command and Data Reordering

The heart of the SDRAM controller is a command and data reordering engine. Command reordering allows banks for future transactions to be opened before the current transaction finishes.

Data reordering allows transactions to be serviced in a different order than they were received when that new order allows for improved utilization of the SDRAM bandwidth. Operations to the same bank and row are performed in order to ensure that operations which impact the same address preserve the data integrity.

The following figure shows the relative timing for a write/read/write/read command sequence performed in order and then the same command sequence performed with data reordering. Data reordering allows the write and read operations to occur in bursts, without bus turnaround timing delay or bank reassignment.

**Figure 148. Data Reordering Effect**



The SDRAM controller schedules among all pending row and column commands every clock cycle.



#### 7.6.4.2. Bank Policy

The bank policy of the SDRAM controller allows users to request that a transaction's bank remain open after an operation has finished so that future accesses do not delay in activating the same bank and row combination. The controller supports only eight simultaneously-opened banks, so an open bank might get closed if the bank resource is needed for other operations.

Open bank resources are allocated dynamically as SDRAM burst transactions are scheduled. Bank allocation is requested automatically by the controller when an incoming transaction spans multiple SDRAM bursts or by the extended command interface. When a bank must be reallocated, the least-recently-used open bank is used as the replacement.

If the controller determines that the next pending command will cause the bank request to not be honored, the bank might be held open or closed depending on the pending operation. A request to close a bank with a pending operation in the timer bank pool to the same row address causes the bank to remain open. A request to leave a bank open with a pending command to the same bank but a different row address causes a precharge operation to occur.

#### 7.6.4.3. Write Combining

The SDRAM controller combines write operations from successive bursts on a port where the starting address of the second burst is one greater than the ending address of the first burst and the resulting burst length does not overflow the 11-bit burst-length counters.

Write combining does not occur if the previous bus command has finished execution before the new command has been received.

#### 7.6.4.4. Burst Length Support

The controller supports burst lengths of 2, 4, 8, and 16. Data widths of 8, 16, and 32 bits are supported for non-ECC operation and data widths of 24 and 40 bits are supported for operations with ECC enabled. The following table shows the type of SDRAM for each burst length.

**Table 69. SDRAM Burst Lengths**

| Burst Length | SDRAM              |
|--------------|--------------------|
| 4            | LPDDR2, DDR2       |
| 8            | DDR2, DDR3, LPDDR2 |
| 16           | LPDDR2             |

#### Width Matching

The SDRAM controller automatically performs data width conversion.



#### 7.6.4.5. ECC

The single-port controller supports memory ECC calculated by the controller. The controller ECC employs standard Hamming logic to detect and correct single-bit errors and detect double-bit errors. The controller ECC is available for 16-bit and 32-bit widths, each requiring an additional 8 bits of memory, resulting in an actual memory width of 24-bits and 40-bits, respectively.

**Note:** The level of ECC support is package dependent.

Functions the controller ECC provides are:

- Byte Writes
- ECC Write Backs
- Notification of ECC Errors

##### 7.6.4.5.1. Byte Writes

The memory controller performs a read-modify-write operation to ensure that the ECC data remains valid when a subset of the bits of a word is being written.

Byte writes with ECC enabled are executed as a read-modify-write. Typical operations only use a single entry in the timer bank pool. Controller ECC enabled sub-word writes use two entries. The first operation is a read and the second operation is a write. These two operations are transferred to the timer bank pool with an address dependency so that the write cannot be performed until the read data has returned. This approach ensures that any subsequent operations to the same address (from the same port) are executed after the write operation, because they are ordered on the row list after the write operation.

If an entire word is being written (but less than a full burst) and the DM pins are connected, no read is necessary and only that word is updated. If controller ECC is disabled, byte-writes have no performance impact.

##### 7.6.4.5.2. ECC Write Backs

If the controller ECC is enabled and a read operation results in a correctable ECC error, the controller corrects the location in memory, if write backs are enabled. The correction results in scheduling a new read-modify-write.

A new read is performed at the location to ensure that a write operation modifying the location is not overwritten. The actual ECC correction operation is performed as a read-modify-write operation. ECC write backs are enabled and disabled through the `cfg_enable_ecc_code_overwrites` field in the `ctrlcfg` register.

**Note:** Double-bit errors do not generate read-modify-write commands. Instead, double-bit error address and count are reported through the `erraddr` and `dbeccount` registers, respectively. In addition, a double-bit error interrupt can be enabled through the `dramintr` register.



#### 7.6.4.5.3. User Notification of ECC Errors

When an ECC error occurs, an interrupt signal notifies the MPU subsystem, and the ECC error information is stored in the status registers. The memory controller provides interrupts for single-bit and double-bit errors.

The status of interrupts and errors are recorded in status registers, as follows:

- The `dramsts` register records interrupt status.
- The `dramintr` register records interrupt masks.
- The `sbecount` register records the single-bit error count.
- The `dbecount` register records the double-bit error count.
- The `erraddr` register records the address of the most recent error.

For a 32-bit interface, ECC is calculated across a span of 8 bytes, meaning the error address is a multiple of 8 bytes (4-bytes\*2 burst length). To find the byte address of the word that contains the error, you must multiply the value in the `erraddr` register by 8.

#### 7.6.4.6. Interleaving Options

The controller supports the following address-interleaving options:

- Non-interleaved
- Bank interleave without chip select interleave
- Bank interleave with chip select interleave

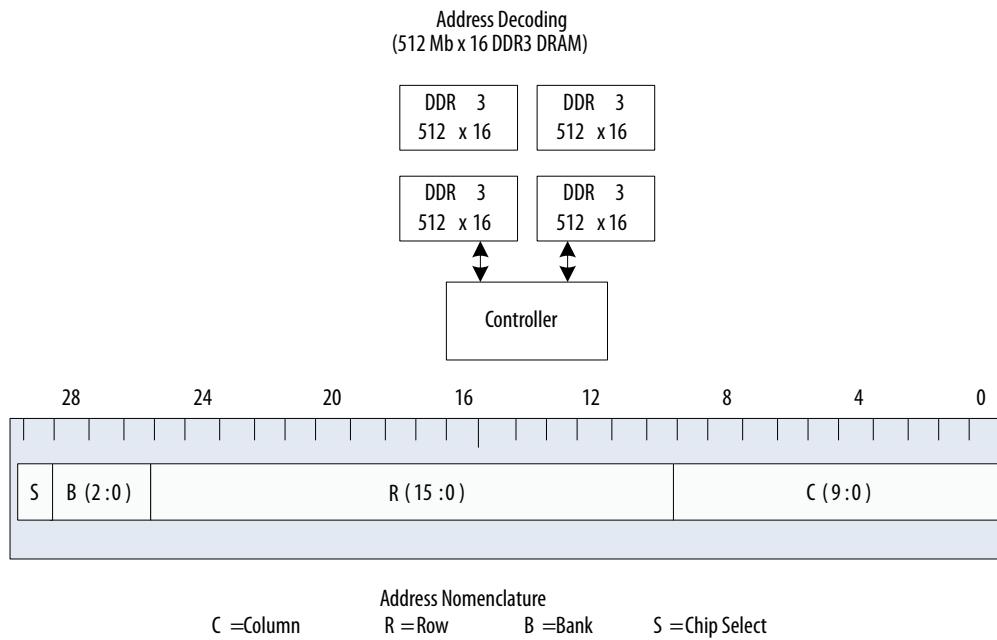
The following interleaving examples use 512 megabits (Mb) x 16 DDR3 chips and are documented as byte addresses. For RAMs with smaller address fields, the order of the fields stays the same but the widths may change.

##### Non-interleaved

RAM mapping is non-interleaved.



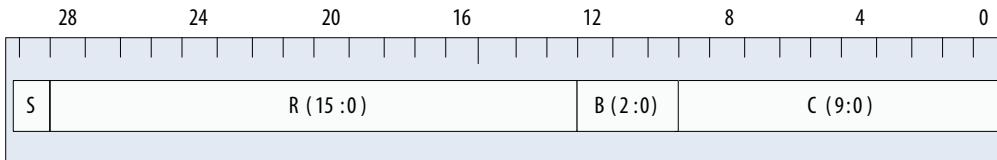
**Figure 149. Non-interleaved Address Decoding**



#### Bank Interleave Without Chip Select Interleave

Bank interleave without chip select interleave swaps row and bank from the non-interleaved address mapping. This interleaving allows smaller data structures to spread across all banks in a chip.

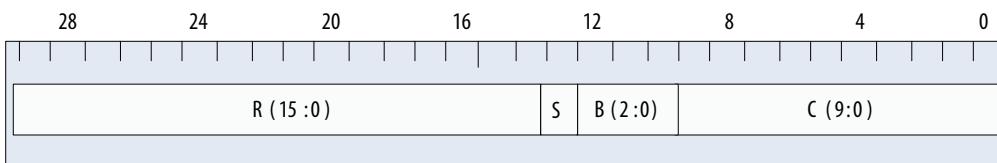
**Figure 150. Bank Interleave Without Chip Select Interleave Address Decoding**



#### Bank Interleave with Chip Select Interleave

Bank interleave with chip select interleave moves the row address to the top, followed by chip select, then bank, and finally column address. This interleaving allows smaller data structures to spread across multiple banks and chips (giving access to 16 total banks for multithreaded access to blocks of memory). Memory timing is degraded when switching between chips.

**Figure 151. Bank Interleave With Chip Select Interleave Address Decoding**





#### 7.6.4.7. AXI-Exclusive Support

The single-port controller supports AXI-exclusive operations. The controller implements a table shared across all masters, which can store up to 16 pending writes. Table entries are allocated on an exclusive read and table entries are deallocated on a successful write to the same address by any master.

Any exclusive write operation that is not present in the table returns an exclusive fail as acknowledgement to the operation. If the table is full when the exclusive read is performed, the table replaces a random entry.

**Note:** When using AXI-exclusive operations, accessing the same location from Avalon-MM interfaces can result in unpredictable results.

#### 7.6.4.8. Memory Protection

The single-port controller has address protection to allow the software to configure basic protection of memory from all masters in the system. If the system has been designed exclusively with AMBA masters, TrustZone® is supported. Ports that use Avalon-MM can be configured for port level protection.

Memory protection is based on physical addresses in memory. The single-port controller can configure up to 20 rules to allow or prevent masters from accessing a range of memory based on their AxIDs, level of security and the memory region being accessed. If no rules are matched in an access, then default settings take effect.

The rules are stored in an internal protection table and can be accessed through indirect addressing offsets in the `protruleaddr` register in the CSR. To read a specific rule, set the `readrule` bit and write the appropriate offset in the `ruleoffset` field of the `protruleaddr` register.

To write a new rule, three registers in the CSR must be configured:

1. The `protportdefault` register is programmed to control the default behavior of memory accesses when no rules match. When a bit is clear, all default accesses from that port pass. When a bit is set, all default accesses from that port fails. The bits are assigned as follows:

**Table 70. `protportdefault` register**

| Bits  | Description  |
|-------|--|
| 31:10 | reserved   |
| 9     | When this bit is set to 1, deny CPU writes during a default transaction.<br>When this bit is clear, allow CPU writes during a default transaction. |
| 8     | When this bit is set to 1, deny L3 writes during a default transaction.<br>When this bit is clear, allow L3 writes during a default transaction.   |
| 7     | When this bit is set to 1, deny CPU reads during a default transaction.<br>When this bit is clear, allow CPU reads during a default transaction.   |
| 6     | When this bit is set to 1, deny L3 reads during a default transaction.<br>When this bit is clear, allow L3 reads during a default transaction.     |
| 5:0   | When this bit is set to 1, deny accesses from FPGA-to-SDRAM ports 0 through 5 during a default transaction.  |

*continued...*



| Bits | Description   |
|------|---|
|      | When this bit is clear, allow accesses from FPGA-to-SDRAM ports 0 through 5 during a default transaction. |

2. The `protruleid` register gives the bounds of the AxID value that allows an access
3. The `protruledata` register configures the specific security characteristics for a rule.

Once the registers are configured, they can be committed to the internal protection table by programming the `ruleoffset` field and setting the `writerule` bit in the `protruledwr` register.

Secure and non-secure regions are specified by rules containing a starting address and ending address with 1 MB boundaries for both addresses. You can override the port defaults and allow or disallow all transactions.

The following table lists the fields that you can specify for each rule.

**Table 71. Fields for Rules in Memory Protection Table**

| Field                    | Width | Description  |
|--------------------------|-------|--|
| Valid                    | 1     | Set to 1 to activate the rule. Set to 0 to deactivate the rule.  |
| Port Mask <sup>(7)</sup> | 10    | Specifies the set of ports to which the rule applies, with one bit representing each port, as follows: bits 0 to 5 correspond to FPGA fabric ports 0 to 5, bit 6 corresponds to AXI L3 interconnect read, bit 7 is the CPU read, bit 8 is L3 interconnect write, and bit 9 is the CPU write. |
| AxID_low <sup>(7)</sup>  | 12    | Low transfer AxID of the rules to which this rule applies. Incoming transactions match if they are greater than or equal to this value. Ports with smaller AxIDs have the AxID shifted to the lower bits and zero padded at the top.   |
| AxID_high <sup>(7)</sup> | 12    | High transfer AxID of the rules to which this rule applies. Incoming transactions match if they are less than or equal to this value.  |
| Address_low              | 12    | Points to a 1MB block and is the lower address. Incoming addresses match if they are greater than or equal to this value.  |
| Address_high             | 12    | Upper limit of address. Incoming addresses match if they are less than or equal to this value.   |
| Protection               | 2     | A value of 0x0 indicates that the rule applies to secure transactions; a value of 0x1 indicates the rule applies to non-secure transactions. Values 0x2 and 0x3 set the region to shared, meaning both secure and non-secure accesses are valid.   |
| Fail/allow               | 1     | Set this value to 1 to force the operation to fail or succeed.   |

Each port has a default access status of either allow or fail. Rules with the opposite allow/fail value can override the default. The system evaluates each transaction against every rule in the memory protection table. If a transaction arrives at a port that defaults to access allowed, it fails only if a rule with the fail bit matches the

---

<sup>(7)</sup> Although AxID and Port Mask could be redundant, including both in the table allows possible compression of rules. If masters connected to a port do not have contiguous AxIDs, a port-based rule might be more efficient than an AxID-based rule, in terms of the number of rules needed.



transaction. Conversely, if a transaction arrives at a port that has the default rule set to access denied, it allows access only if there is a matching rule that forces accessed allowed. Transactions that fail the protection rules return a slave error (SLVERR).

The recommended sequence for writing a rule is:

1. Write the `protruledwr` register fields as follows:
  - `ruleoffset` = offset selected by user that points to indirect offset in an internal protection table..
  - `writerule` = 0
  - `readrule` = 0
2. Write the `proruleaddr`, `proruleid`, and `proruledata` registers so you configure the rule you would like to enforce.
3. Write the `protruledwr` register fields as follows:
  - `ruleoffset` = offset of the rule that needs to be written
  - `writerule` = 1
  - `readrule` = 0

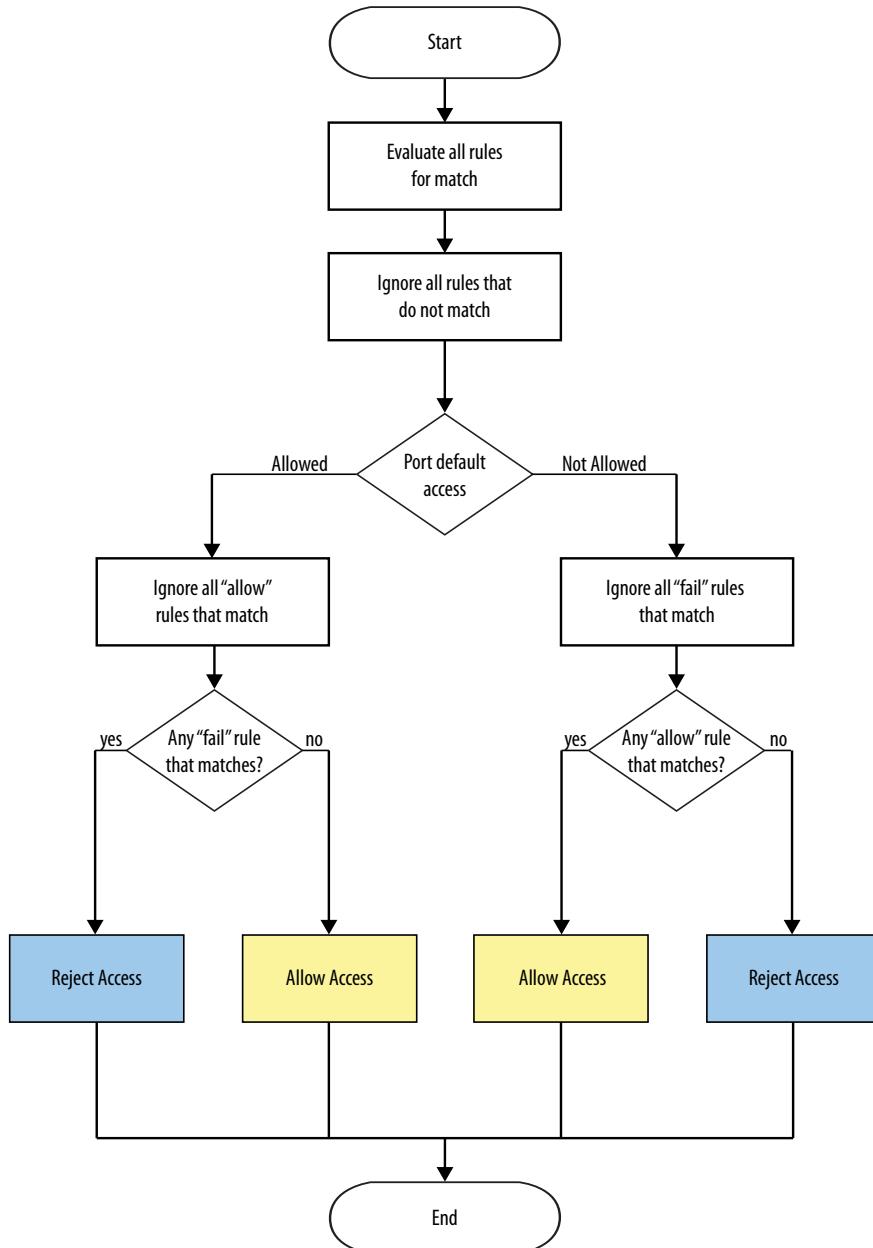
Similarly, the recommended sequence for reading a rule is:

1. Write the `protruledwr` register fields as follows:
  - `ruleoffset` = offset of the rule that needs to be written
  - `writerule` = 0
  - `readrule` = 0
2. Write the `protruledwr` register fields as follows:
  - `ruleoffset` = offset of the rule that needs to be read
  - `writerule` = 0
  - `readrule` = 1
3. Read the values of the `proruleaddr`, `proruleid`, and `proruledata` registers to determine the rule parameters.

The following figure represents an overview of how the protection rules are applied. There is no priority among the 20 rules. All rules are always evaluated in parallel.



**Figure 152. SDRAM Protection Access Flow Diagram**



Exclusive transactions are security checked on the read operation only. A write operation can occur only if a valid read is marked in the internal exclusive table. Consequently, a master performing an exclusive read followed by a write, can write to memory only if the exclusive read was successful.

### Related Information

#### ARM TrustZone®

For more information about TrustZone® refer to the ARM web page.

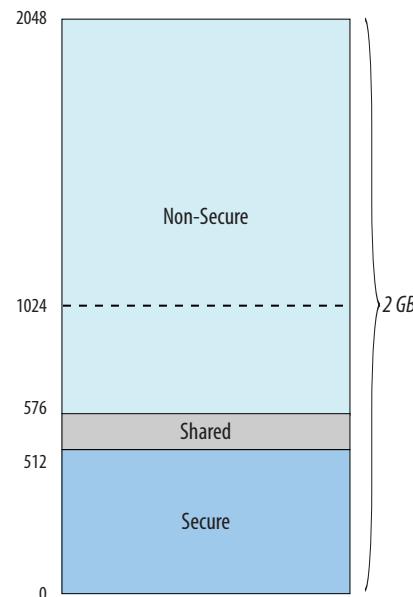
#### 7.6.4.9. Example of Configuration for TrustZone

For a TrustZone® configuration, memory is divided into a range of memory accessible by secure masters and a range of memory accessible by non-secure masters. The two memory address ranges may have a range of memory that overlaps.

This example implements the following memory configuration:

- 2 GB total RAM size
- 0–512 MB dedicated secure area
- 513–576 MB shared area
- 577–2048 MB dedicated non-secure area

**Figure 153. Example Memory Configuration**



In this example, each port is configured by default to disallow all accesses. The following table shows the two rules programmed into the memory protection table.

**Table 72. Rules in Memory Protection Table for Example Configuration**

| Rule # | Port Mask    | AxID Low | AxID High     | Address Low | Address High | protruledata.security | Fail/Allow |
|--------|--------------|----------|---------------|-------------|--------------|-----------------------|------------|
| 1      | 0x3FF (1023) | 0x000    | 0xFFFF (4095) | 0           | 576          | 0x1                   | Allow      |
| 2      | 0x3FF (1023) | 0x000    | 0xFFFF (4095) | 512         | 2047         | 0x0                   | Allow      |

The port mask value, AxID Low, and AxID High, apply to all ports and all transfers within those ports. Each access request is evaluated against the memory protection table, and will fail unless there is a rule match allowing a transaction to complete successfully.



**Table 73. Result for a Sample Set of Transactions**

| Operation | Source              | Address Accesses | Security Access Type | Result | Comments   |
|-----------|---------------------|------------------|----------------------|--------|--|
| Read      | CPU                 | 4096             | secure               | Allow  | Matches rule 1.  |
| Write     | CPU                 | 536, 870, 912    | secure               | Allow  | Matches rule 1.  |
| Write     | L3 attached masters | 605, 028, 350    | secure               | Fail   | Does not match rule 1 (out of range of the address field), does not match rule 2 (protection bit incorrect). |
| Read      | L3 attached masters | 4096             | non-secure           | Fail   | Does not match rule 1 (AxPROT signal value wrong), does not match rule 2 (not in address range).             |
| Write     | CPU                 | 536, 870, 912    | non-secure           | Allow  | Matches rule 2.  |
| Write     | L3 attached masters | 605, 028, 350    | non-secure           | Allow  | Matches rule 2.  |

**Note:** If a master is using the Accelerator Coherency Port (ACP) to maintain cache coherency with the Cortex-A9 MPCore processor, then the address ranges in the rules of the memory protection table should be made mutually exclusive, such that the secure and non-secure regions do not overlap and any area that is shared is part of the non-secure region. This configuration prevents coherency issues from occurring.

## 7.7. SDRAM Power Management

The SDRAM controller subsystem supports the following power saving features in the SDRAM:

- Partial array self-refresh (PASR)
- Power down
- Deep power down for LPDDR2

To enable self-refresh for the memories of one or both chip selects, program the selfshreq bit and the sefrfshmask bit in the lowpwreq register.

Power-saving mode initiates either due to a user command or from inactivity. The number of idle clock cycles after which a memory can be put into power-down mode is programmed through the autopdycycles field of the lowpwrtiming register.

Power-down mode forces the SDRAM burst-scheduling bank-management logic to close all banks and issue the power down command. The SDRAM automatically reactivates when an active SDRAM command is received.

To enable deep power down request for the LPDDR2 memories of one or both chip selects, program the deppwrndnreq bit and the deepwrdnmask field of the lowpwreq register.

Other power-down modes are performed only under user control.



## 7.8. DDR PHY

The DDR PHY connects the memory controller and external memory devices in the speed critical command path.

The DDR PHY implements the following functions:

- Calibration—the DDR PHY supports the JEDEC-specified steps to synchronize the memory timing between the controller and the SDRAM chips. The calibration algorithm is implemented in software.
- Memory device initialization—the DDR PHY performs the mode register write operations to initialize the devices. The DDR PHY handles re-initialization after a deep power down.
- Single-data-rate to double-data-rate conversion.

## 7.9. Clocks

All clocks are assumed to be asynchronous with respect to the `ddr_dqs_clk` memory clock. All transactions are synchronized to memory clock domain.

**Table 74. SDRAM Controller Subsystem Clock Domains**

| Clock Name                      | Description   |
|---------------------------------|---|
| <code>ddr_dq_clk</code>         | Clock for PHY   |
| <code>ddr_dqs_clk</code>        | Clock for MPFE, single-port controller, CSR access, and PHY                 |
| <code>ddr_2x_dqs_clk</code>     | Clock for PHY that provides up to 2 times <code>ddr_dq_clk</code> frequency |
| <code>l4_sp_clk</code>          | Clock for CSR interface   |
| <code>mpu_12_ram_clk</code>     | Clock for MPU interface   |
| <code>l3_main_clk</code>        | Clock for L3 interface  |
| <code>f2h_sdram_clk[5:0]</code> | Six separate clocks used for the FPGA-to-HPS SDRAM ports to the FPGA fabric |

In terms of clock relationships, the FPGA fabric connects the appropriate clocks to write data, read data, and command ports for the constructed ports.

## 7.10. Resets

The SDRAM controller subsystem supports a full reset (cold reset) and a warm reset. The SDRAM controller can be configured to preserve memory contents during a warm reset.

To preserve memory contents, the reset manager can request that the single-port controller place the SDRAM in self-refresh mode prior to issuing the warm reset. If self-refresh mode is enabled before the warm reset to preserve memory contents, the PHY and the memory timing logic is not reset, but the rest of the controller is reset.



### 7.10.1. Taking the SDRAM Controller Subsystem Out of Reset

When a cold or warm reset is issued in the HPS, the Reset Manager resets this module and holds it in reset until software releases it.

After the MPU boots up, it can deassert the reset signal by clearing the appropriate bits in the Reset Manager's corresponding reset trigger.

### 7.11. Port Mappings

The memory interface controller has a set of command, read data, and write data ports that support AXI3, AXI4 and Avalon-MM. Tables are provided to identify port assignments and functions.

**Table 75. Command Port Assignments**

| Command Port | Allowed Functions  |
|--------------|--|
| 0, 2, 4      | FPGA fabric AXI read command ports<br>FPGA fabric Avalon-MM read or write command ports  |
| 1, 3, 5      | FPGA fabric AXI write command ports<br>FPGA fabric Avalon-MM read or write command ports |
| 6            | L3 AXI read command port   |
| 7            | MPU AXI read command port  |
| 8            | L3 AXI write command port  |
| 9            | MPU AXI write command port   |

**Table 76. Read Port Assignments**

| Read Port  | Allowed Functions  |
|------------|--|
| 0, 1, 2, 3 | 64-bit read data from the FPGA fabric. When 128-bit data read ports are created, then read data ports 0 and 1 get paired as well as 2 and 3. |
| 4          | 32-bit L3 read data port   |
| 5          | 64-bit MPU read data port  |

**Table 77. Write Port Assignments**

| Write Port | Allowed Functions   |
|------------|---|
| 0, 1, 2, 3 | 64-bit write data from the FPGA fabric. When 128-bit data write ports are created, then write data ports 0 and 1 get paired as well as 2 and 3. |
| 4          | 32-bit L3 write data port   |
| 5          | 64-bit MPU write data port  |

### 7.12. Initialization

The SDRAM controller subsystem has control and status registers (CSRs) which control the operation of the controller including DRAM type, DRAM timing parameters and relative port priorities. It also has a small set of bits which depend on the FPGA fabric to configure ports between the memory controller and the FPGA fabric; these bits are set for you when you configure your implementation using the HPS GUI in Qsys.



The CSRs are configured using a dedicated slave interface, which provides access to the registers. This region controls all SDRAM operation, MPFE scheduler configuration, and PHY calibration.

The FPGA fabric interface configuration is programmed into the FPGA fabric and the values of these register bits can be read by software. The ports can be configured without software developers needing to know how the FPGA-to-HPS SDRAM interface has been configured.

### 7.12.1. FPGA-to-SDRAM Protocol Details

The following topics summarize signals for the Avalon-MM Bidirectional port, Avalon-MM Write Port, Avalon-MM Read Port, and AXI port.

**Note:** If your device has multiple FPGA hardware images, then the same FPGA-to-SDRAM port configuration should be used across all designs.

#### 7.12.1.1. Avalon-MM Bidirectional Port

The Avalon-MM bidirectional ports are standard Avalon-MM ports used to dispatch read and write operations.

Each configured Avalon-MM bidirectional port consists of the signals listed in the following table.

**Table 78. Avalon-MM Bidirectional Port Signals**

| Name          | Bit Width           | Input/Output Direction | Function  |
|---------------|---------------------|------------------------|---|
| clk           | 1                   | In                     | Clock for the Avalon-MM interface   |
| read          | 1                   | In                     | Indicates read transaction <sup>(8)</sup>   |
| write         | 1                   | In                     | Indicates write transaction <sup>(8)</sup>  |
| address       | 32                  | In                     | Address of the transaction  |
| readdata      | 32, 64, 128, or 256 | Out                    | Read data return  |
| readdatavalid | 1                   | Out                    | Indicates the readdata signal contains valid data in response to a previous read request.     |
| writedata     | 32, 64, 128, or 256 | In                     | Write data for a transaction  |
| byteenable    | 4, 8, 16, 32        | In                     | Byte enables for each write byte lane   |
| waitrequest   | 1                   | Out                    | Indicates need for additional cycles to complete a transaction                                |
| burstcount    | 11                  | In                     | Transaction burst length. The value of the maximum burstcount parameter must be a power of 2. |

<sup>(8)</sup> The Avalon-MM protocol does not allow read and write transactions to be posted concurrently.



The read and write interfaces are configured to the same size. The byte-enable size scales with the data bus size.

#### Related Information

##### [Avalon Interface Specifications](#)

Information about the Avalon-MM protocol

### 7.12.1.2. Avalon-MM Write-Only Port

The Avalon-MM write-only ports are standard Avalon-MM ports used to dispatch write operations. Each configured Avalon-MM write port consists of the signals listed in the following table.

**Table 79. Avalon-MM Write-Only Port Signals**

| Name        | Bits                | Direction | Function   |
|-------------|---------------------|-----------|--|
| reset       | 1                   | In        | Reset  |
| clk         | 1                   | In        | Clock  |
| write       | 1                   | In        | Indicates write transaction                                    |
| address     | 32                  | In        | Address of the transaction                                     |
| writedata   | 32, 64, 128, or 256 | In        | Write data for a transaction                                   |
| byteenable  | 4, 8, 16, 32        | In        | Byte enables for each write byte                               |
| waitrequest | 1                   | Out       | Indicates need for additional cycles to complete a transaction |
| burstcount  | 11                  | In        | Transaction burst length                                       |

#### Related Information

##### [Avalon Interface Specifications](#)

Information about the Avalon-MM protocol

### 7.12.1.3. Avalon-MM Read Port

The Avalon-MM read ports are standard Avalon-MM ports used only to dispatch read operations. Each configured Avalon-MM read port consists of the signals listed in the following table.

**Table 80. Avalon-MM Read Port Signals**

| Name    | Bits | Direction | Function                   |
|---------|------|-----------|----------------------------|
| reset   | 1    | In        | Reset                      |
| clk     | 1    | In        | Clock                      |
| read    | 1    | In        | Indicates read transaction |
| address | 32   | In        | Address of the transaction |

*continued...*



| Name          | Bits                | Direction | Function  |
|---------------|---------------------|-----------|---|
| readdata      | 32, 64, 128, or 256 | Out       | Read data return  |
| readdatavalid | 1                   | Out       | Flags valid cycles for read data return   |
| waitrequest   | 1                   | Out       | Indicates the need for additional cycles to complete a transaction. Needed for read operations when delay is needed to accept the read command. |
| burstcount    | 11                  | In        | Transaction burst length  |

### Related Information

#### Avalon Interface Specifications

Information about the Avalon-MM protocol

#### 7.12.1.4. AXI Port

The AXI port uses an AXI-3 interface. Each configured AXI port consists of the signals listed in the following table. Because the AXI protocol allows simultaneous read and write commands to be issued, two SDRAM control ports are required to form an AXI interface.

**Table 81. AXI Port Signals**

| Name    | Bits               | Direction | Channel       | Function   |
|---------|--------------------|-----------|---------------|--|
| ARESETn | 1                  | In        | n/a           | Reset  |
| ACLK    | 1                  | In        | n/a           | Clock  |
| AWID    | 4                  | In        | Write address | Write identification tag   |
| AWADDR  | 32                 | In        | Write address | Write address  |
| AWLEN   | 4                  | In        | Write address | Write burst length   |
| AWSIZE  | 3                  | In        | Write address | Width of the transfer size   |
| AWBURST | 2                  | In        | Write address | Burst type   |
| AWLOCK  | 2                  | In        | Write address | Lock type signal which indicates if the access is exclusive; valid values are 0x0 (normal access) and 0x1 (exclusive access) |
| AWCACHE | 4                  | In        | Write address | Cache policy type  |
| AWPROT  | 3                  | In        | Write address | Protection-type signal used to indicate whether a transaction is secure or non-secure  |
| AWREADY | 1                  | Out       | Write address | Indicates ready for a write command  |
| AWVALID | 1                  | In        | Write address | Indicates valid write command.   |
| WID     | 4                  | In        | Write data    | Write data transfer ID   |
| WDATA   | 32, 64, 128 or 256 | In        | Write data    | Write data   |
| WSTRB   | 4, 8, 16, 32       | In        | Write data    | Byte-based write data strobe. Each bit width corresponds to 8 bit wide transfer for 32-bit wide to 256-bit wide transfer.    |

*continued...*



| Name    | Bits                     | Direction | Channel        | Function   |
|---------|--------------------------|-----------|----------------|--|
| WLAST   | 1                        | In        | Write data     | Last transfer in a burst   |
| WVALID  | 1                        | In        | Write data     | Indicates write data and strobes are valid   |
| WREADY  | 1                        | Out       | Write data     | Indicates ready for write data and strobes   |
| BID     | 4                        | Out       | Write response | Write response transfer ID   |
| BRESP   | 2                        | Out       | Write response | Write response status  |
| BVALID  | 1                        | Out       | Write response | Write response valid signal  |
| BREADY  | 1                        | In        | Write response | Write response ready signal  |
| ARID    | 4                        | In        | Read address   | Read identification tag  |
| ARADDR  | 32                       | In        | Read address   | Read address   |
| ARLEN   | 4                        | In        | Read address   | Read burst length  |
| ARSIZE  | 3                        | In        | Read address   | Width of the transfer size   |
| ARBURST | 2                        | In        | Read address   | Burst type   |
| ARLOCK  | 2                        | In        | Read address   | Lock type signal which indicates if the access is exclusive; valid values are 0x0 (normal access) and 0x1 (exclusive access) |
| ARCACHE | 4                        | In        | Read address   | Lock type signal which indicates if the access is exclusive; valid values are 0x0 (normal access) and 0x1 (exclusive access) |
| ARPROT  | 3                        | In        | Read address   | Protection-type signal used to indicate whether a transaction is secure or non-secure  |
| ARREADY | 1                        | Out       | Read address   | Indicates ready for a read command   |
| ARVALID | 1                        | In        | Read address   | Indicates valid read command   |
| RID     | 4                        | Out       | Read data      | Read data transfer ID  |
| RDATA   | 32, 64,<br>128 or<br>256 | Out       | Read data      | Read data  |
| RRESP   | 2                        | Out       | Read data      | Read response status   |
| RLAST   | 1                        | Out       | Read data      | Last transfer in a burst   |
| RVALID  | 1                        | Out       | Read data      | Indicates read data is valid   |
| RREADY  | 1                        | In        | Read data      | Read data channel ready signal   |

### Related Information

[ARM AMBA Open Specification](#)

AMBA Open Specifications, including information about the AXI-3 interface

## 7.13. SDRAM Controller Subsystem Programming Model

SDRAM controller configuration occurs through software programming of the configuration registers using the CSR interface.



### 7.13.1. HPS Memory Interface Architecture

The configuration and initialization of the memory interface by the ARM processor is a significant difference compared to the FPGA memory interfaces, and results in several key differences in the way the HPS memory interface is defined and configured.

Boot-up configuration of the HPS memory interface is handled by the initial software boot code, not by the FPGA programmer, as is the case for the FPGA memory interfaces. The Quartus Prime software is involved in defining the configuration of I/O ports which is used by the boot-up code, as well as timing analysis of the memory interface. Therefore, the memory interface must be configured with the correct PHY-level timing information. Although configuration of the memory interface in Qsys is still necessary, it is limited to PHY- and board-level settings.

### 7.13.2. HPS Memory Interface Configuration

To configure the external memory interface components of the HPS, open the HPS interface by selecting the Arria V/Cyclone V Hard Processor System component in Qsys. Within the HPS interface, select the EMIF tab to open the EMIF parameter editor.

The EMIF parameter editor contains four additional tabs: PHY Settings, Memory Parameters, Memory Timing, and Board Settings. The parameters available on these tabs are similar to those available in the parameter editors for non-SoC device families.

There are significant differences between the EMIF parameter editor for the Hard Processor System and the parameter editors for non-SoC devices, as follows:

- Because the HPS memory controller is not configurable through the Quartus Prime software, the Controller and Diagnostic tabs, which exist for non-SoC devices, are not present in the EMIF parameter editor for the hard processor system.
- Unlike the protocol-specific parameter editors for non-SoC devices, the EMIF parameter editor for the Hard Processor System supports multiple protocols, therefore there is an SDRAM Protocol parameter, where you can specify your external memory interface protocol. By default, the EMIF parameter editor assumes the DDR3 protocol, and other parameters are automatically populated with DDR3-appropriate values. If you select a protocol other than DDR3, change other associated parameter values appropriately.
- Unlike the memory interface clocks in the FPGA, the memory interface clocks for the HPS are initialized by the boot-up code using values provided by the configuration process. You may accept the values provided by UniPHY, or you may use your own PLL settings. If you choose to specify your own PLL settings, you must indicate that the clock frequency that UniPHY should use is the requested clock frequency, and not the achieved clock frequency calculated by UniPHY.

**Note:** The HPS does not support EMIF synthesis generation, compilation, or timing analysis. The HPS hard memory controller cannot be bonded with another hard memory controller on the FPGA portion of the device.

### 7.13.3. HPS Memory Interface Simulation

Qsys provides a complete simulation model of the HPS memory interface controller and PHY, providing cycle-level accuracy, comparable to the simulation models for the FPGA memory interface.



The simulation model supports only the skip-cal simulation mode; quick-cal and full-cal are not supported. An example design is not provided. However, you can create a test design by adding the traffic generator component to your design using Qsys. Also, the HPS simulation model does not use external memory pins to connect to the DDR memory model; instead, the memory model is incorporated directly into the HPS SDRAM interface simulation modules. The memory instance incorporated into the HPS model is in the simulation model hierarchy at: hps\_0/fpga\_interfaces/f2sdram/hps\_sdram\_inst/mem/

Simulation of the FPGA-to-SDRAM interfaces requires that you first bring the interfaces out of reset, otherwise transactions cannot occur. Connect the H2F reset to the F2S port resets and add a stage to your testbench to assert and deassert the H2F reset in the HPS. Appropriate Verilog code is shown below:

```
initial
begin
    // Assert reset
    <base name>.hps.fpga_interfaces.h2f_reset_inst.reset_assert();
    // Delay
    #1
    // Deassert reset
    <base name>.hps.fpga_interfaces.h2f_reset_inst.reset_deassert();
end
```

#### 7.13.4. Generating a Preloader Image for HPS with EMIF

To generate a Preloader image for an HPS-based external memory interface, you must complete the following tasks:

- Create a Qsys project.
- Create a top-level file and add constraints.
- Create a Preloader BSP file.
- Create a Preloader image.

##### 7.13.4.1. Creating a Qsys Project

Before you can generate a preloader image, you must create a Qsys project, as follows:

1. On the **Tools** menu in the Quartus Prime software, click **Qsys**.
2. Under **Component library**, expand **Embedded Processor System**, select **Hard Processor System** and click **Add**.
3. Specify parameters for the **FPGA Interfaces**, **Peripheral Pin Multiplexing**, and **HPS Clocks**, based on your design requirements.
4. On the **SDRAM** tab, select the SDRAM protocol for your interface.
5. Populate the necessary parameter fields on the **PHY Settings**, **Memory Parameters**, **Memory Timing**, and **Board Settings** tabs.
6. Add other Qsys components in your Qsys design and make the appropriate bus connections.
7. Save the Qsys project.
8. Click **Generate** on the **Generation** tab, to generate the Qsys design.



#### 7.13.4.2. Creating a Top-Level File and Adding Constraints

This topic describes adding your Qsys system to your top-level design and adding constraints to your design.

1. Add your Qsys system to your top-level design.
2. Add the Quartus Prime IP files (.qip) generated in step 2, to your Quartus Prime project.
3. Perform analysis and synthesis on your design.
4. Constrain your EMIF design by running the `<variation_name>_p0_pin_assignments.tcl` pin constraints script.
5. Add other necessary constraints—such as timing constraints, location assignments, and pin I/O standard assignments—for your design.
6. Compile your design to generate an SRAM object file (.sof) and the hardware handoff files necessary for creating a preloader image.

*Note:* You must regenerate the hardware handoff files whenever the HPS configuration changes; for example, due to changes in Peripheral Pin Multiplexing or I/O standard for HPS pins.

#### Related Information

[Intel SoC FPGA Embedded Design Suite User's Guide](#)

For more information on how to create a preloader BSP file and image.

### 7.14. Debugging HPS SDRAM in the Preloader

To assist in debugging your design, tools are available at the preloader stage.

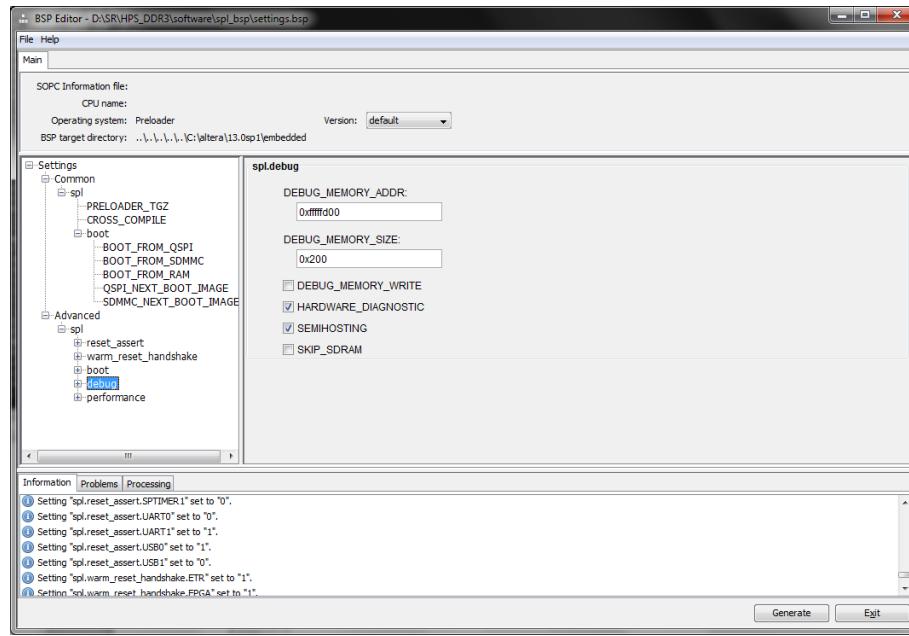
- UART or semihosting printout
- Simple memory test
- Debug report
- Predefined data patterns

The following topics provide procedures for implementing each of the above tools.

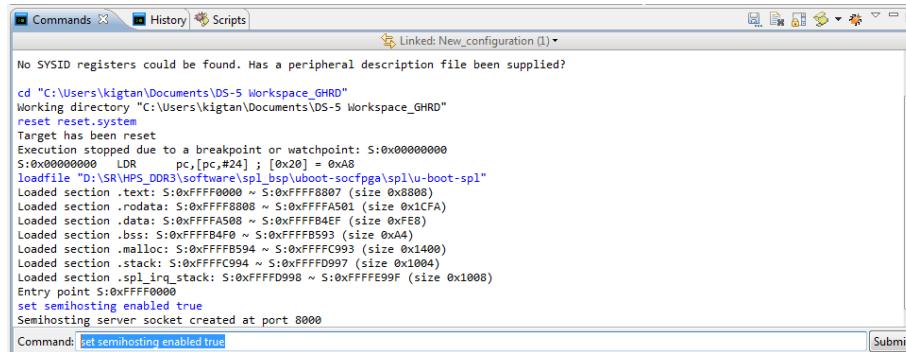
#### 7.14.1. Enabling UART or Semihosting Printout

UART printout is enabled by default. If UART is not available on your system, you can use semihosting together with the debugger tool. To enable semihosting in the Preloader, follow these steps:

1. When you create the .bsp file in the BSP Editor, select **SEMIHOSTING** in the **spl.debug** window.



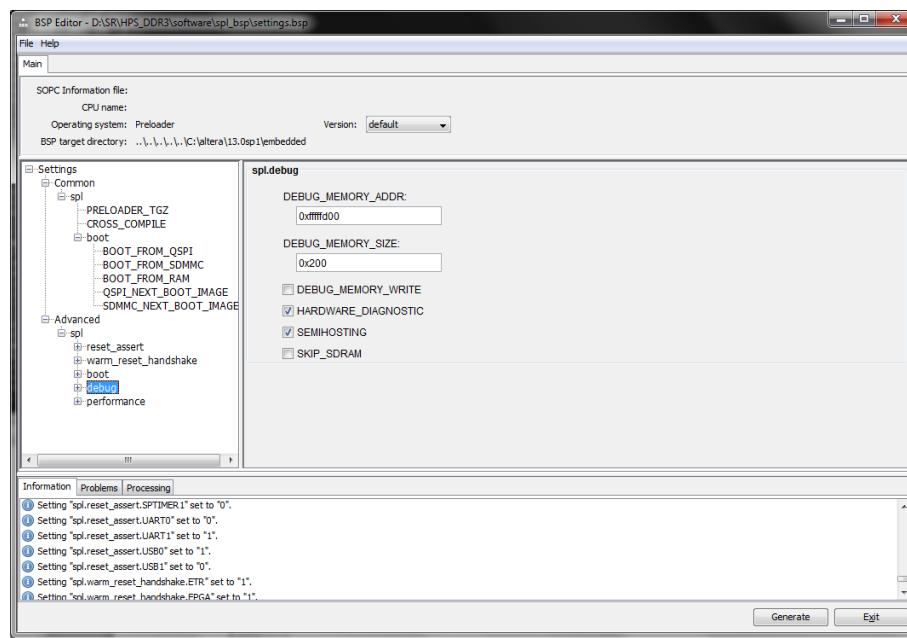
2. Enable semihosting in the debugger, by typing `set semihosting enabled true` at the command line in the debugger.



## 7.14.2. Enabling Simple Memory Test

After the SDRAM is successfully calibrated, a simple memory test may be performed using the debugger.

1. When you create the .bsp file in the BSP Editor, select **HARDWARE\_DIAGNOSTIC** in the **spl.debug** window..



2. The simple memory test assumes SDRAM with a memory size of 1 GB. If your board contains a different SDRAM memory size, open the file `<design folder>\spl_bsp\uboot-socfpga\include\configs\socfpga_cyclone5.h` in a text editor, and change the `PHYS_SDRAM_1_SIZE` parameter at line 292 to specify your actual memory size in bytes.

```

socfpga_cyclone5.h
278     * Hardware drivers
279     */
280
281     /*
282     * SDRAM Memory Map
283     */
284     /* We have 1 bank of DRAM */
285     #define CONFIG_NR_DRAM_BANKS      1
286     /* SDRAM Bank #1 */
287     #define CONFIG_SYS_SDRAM_BASE    0x00000000
288     /* SDRAM memory size */
289     #ifdef CONFIG_SOCFPGA_VIRTUAL_TARGET
290         #define PHYS_SDRAM_1_SIZE        0x80000000
291     #else
292         #define PHYS_SDRAM_1_SIZE        0x40000000
293     #endif
294     /* SDRAM Bank #1 base address */
295     #define PHYS_SDRAM_1             CONFIG_SYS_SDRAM_BASE
296     /* memtest setup */
297     /* Begin and end addresses of the area used by the simple memory test.c */
298     #define CONFIG_SYS_MEMTEST_START  0x00000000
299     #define CONFIG_SYS_MEMTEST_END   PHYS_SDRAM_1_SIZE
300

```

3. The simple memory test assumes SDRAM with a memory size of 1 GB. If your board contains a different SDRAM memory size, open the file `<design folder>\spl_bsp\uboot-socfpga\include\configs\socfpga_arria5.h` in a text editor, and change the `PHYS_SDRAM_1_SIZE` parameter at line 292 to specify your actual memory size in bytes.



```
 socfga_cyclone5.h
278  * Hardware drivers
279  */
280
281  /*
282  * SDRAM Memory Map
283  */
284  /* We have 1 bank of DRAM */
285  #define CONFIG_NR_DRAM_BANKS      1
286  /* SDRAM Bank #1 */
287  #define CONFIG_SYS_SDRAM_BASE     0x00000000
288  /* SDRAM memory size */
289  #ifndef CONFIG_SOCFPGA_VIRTUAL_TARGET
290  #define PHYS_SDRAM_1_SIZE        0x80000000
291  #else
292  #define PHYS_SDRAM_1_SIZE        0x40000000
293  #endif
294  /* SDRAM Bank #1 base address */
295  #define PHYS_SDRAM_1             CONFIG_SYS_SDRAM_BASE
296  /* memtest setup */
297  /* Begin and end addresses of the area used by the simple memory test.c */
298  #define CONFIG_SYS_MEMTEST_START  0x00000000
299  #define CONFIG_SYS_MEMTEST_END    PHYS_SDRAM_1_SIZE
300
```

### 7.14.3. Enabling the Debug Report

You can enable the SDRAM calibration sequencer to produce a debug report on the UART printout or semihosting output. To enable the debug report, follow these steps:

1. After you have enabled the UART or semihosting, open the file *<project directory>\hps\_isw\_handoff\sequencerDefines.hin* a text editor.
2. Locate the line `#define RUNTIME_CAL_REPORT 0` and change it to `#define RUNTIME_CAL_REPORT 1`.



Figure 154. Semihosting Printout With Debug Support Enabled

```
U-Boot SPL 2012.10 (Sep 09 2013 - 19:25:45)
SDRAM: Initializing PMW registers
SEQ.C: Preparing to start memory calibration
SEQ.C: DQS Enable ; Group 0 ; Rank 0 ; Start VFIFO 5 ; Phase 4 ; Delay 3
SEQ.C: DQS Enable ; Group 0 ; Rank 0 ; End VFIFO 6 ; Phase 4 ; Delay 14
SEQ.C: DQS Enable ; Group 0 ; Rank 0 ; Center VFIFO 6 ; Phase 1 ; Delay 3
SEQ.C: Read Deskew ; DQ 0 ; Rank 0 ; Left edge 17 ; Right edge 27 ; DQ delay 3 ; DQS delay 12
SEQ.C: Read Deskew ; DQ 1 ; Rank 0 ; Left edge 30 ; Right edge 23 ; DQ delay 11 ; DQS delay 12
SEQ.C: Read Deskew ; DQ 2 ; Rank 0 ; Left edge 30 ; Right edge 24 ; DQ delay 11 ; DQS delay 12
SEQ.C: Read Deskew ; DQ 3 ; Rank 0 ; Left edge 30 ; Right edge 24 ; DQ delay 11 ; DQS delay 12
SEQ.C: Read Deskew ; DQ 4 ; Rank 0 ; Left edge 17 ; Right edge 27 ; DQ delay 3 ; DQS delay 12
SEQ.C: Read Deskew ; DQ 5 ; Rank 0 ; Left edge 31 ; Right edge 27 ; DQ delay 3 ; DQS delay 12
SEQ.C: Read Deskew ; DQ 6 ; Rank 0 ; Left edge 31 ; Right edge 27 ; DQ delay 3 ; DQS delay 12
SEQ.C: Read Deskew ; DQ 7 ; Rank 0 ; Left edge 30 ; Right edge 23 ; DQ delay 11 ; DQS delay 12
SEQ.C: Write Deskew ; DQ 0 ; Rank 0 ; Left edge 31 ; Right edge 13 ; DQ delay 9 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 1 ; Rank 0 ; Left edge 29 ; Right edge 14 ; DQ delay 6 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 2 ; Rank 0 ; Left edge 29 ; Right edge 19 ; DQ delay 5 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 3 ; Rank 0 ; Left edge 29 ; Right edge 17 ; DQ delay 6 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 4 ; Rank 0 ; Left edge 31 ; Right edge 14 ; DQ delay 9 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 5 ; Rank 0 ; Left edge 31 ; Right edge 19 ; DQ delay 5 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 6 ; Rank 0 ; Left edge 28 ; Right edge 19 ; DQ delay 5 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 7 ; Rank 0 ; Left edge 31 ; Right edge 17 ; DQ delay 6 ; DQS delay 4
SEQ.C: Read after Write ; DQ 0 ; Rank 0 ; Start VFIFO 5 ; Phase 5 ; Delay 13
SEQ.C: Read after Write ; DQ 1 ; Rank 0 ; Left edge 24 ; Right edge 19 ; DQ delay 3 ; DQS delay 12
SEQ.C: Read after Write ; DQ 2 ; Rank 0 ; Left edge 31 ; Right edge 9 ; DQ delay 11 ; DQS delay 12
SEQ.C: Read after Write ; DQ 3 ; Rank 0 ; Left edge 30 ; Right edge 9 ; DQ delay 11 ; DQS delay 12
SEQ.C: Read after Write ; DQ 4 ; Rank 0 ; Left edge 24 ; Right edge 19 ; DQ delay 3 ; DQS delay 12
SEQ.C: Read after Write ; DQ 5 ; Rank 0 ; Left edge 31 ; Right edge 8 ; DQ delay 12 ; DQS delay 12
SEQ.C: Read after Write ; DQ 6 ; Rank 0 ; Left edge 31 ; Right edge 8 ; DQ delay 12 ; DQS delay 12
SEQ.C: Read after Write ; DQ 7 ; Rank 0 ; Left edge 31 ; Right edge 8 ; DQ delay 12 ; DQS delay 12
SEQ.C: DQS Enable ; Group 1 ; Rank 0 ; Start VFIFO 5 ; Phase 5 ; Delay 13
SEQ.C: DQS Enable ; Group 1 ; Rank 0 ; Center VFIFO 6 ; Phase 6
SEQ.C: DQS Enable ; Group 1 ; Rank 0 ; Left edge 1 ; Right edge 27 ; DQ delay 11
SEQ.C: Read Deskew ; DQ 8 ; Rank 0 ; Left edge 29 ; Right edge 26 ; DQ delay 1 ; DQS delay 11
SEQ.C: Read Deskew ; DQ 9 ; Rank 0 ; Left edge 18 ; Right edge 26 ; DQ delay 0 ; DQS delay 11
SEQ.C: Read Deskew ; DQ 10 ; Rank 0 ; Left edge 30 ; Right edge 24 ; DQ delay 10 ; DQS delay 11
SEQ.C: Read Deskew ; DQ 11 ; Rank 0 ; Left edge 30 ; Right edge 9 ; DQ delay 11 ; DQS delay 11
SEQ.C: Read Deskew ; DQ 12 ; Rank 0 ; Left edge 16 ; Right edge 27 ; DQ delay 1 ; DQS delay 11
SEQ.C: Read Deskew ; DQ 13 ; Rank 0 ; Left edge 31 ; Right edge 16 ; DQ delay 11 ; DQS delay 11
SEQ.C: Read Deskew ; DQ 14 ; Rank 0 ; Left edge 13 ; Right edge 27 ; DQ delay 0 ; DQS delay 11
SEQ.C: Read Deskew ; DQ 15 ; Rank 0 ; Left edge 30 ; Right edge 25 ; DQ delay 9 ; DQS delay 11
SEQ.C: Write Deskew ; DQ 9 ; Rank 0 ; Left edge 31 ; Right edge 16 ; DQ delay 7 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 10 ; Rank 0 ; Left edge 29 ; Right edge 16 ; DQ delay 6 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 11 ; Rank 0 ; Left edge 29 ; Right edge 16 ; DQ delay 6 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 12 ; Rank 0 ; Left edge 31 ; Right edge 16 ; DQ delay 7 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 13 ; Rank 0 ; Left edge 31 ; Right edge 16 ; DQ delay 7 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 14 ; Rank 0 ; Left edge 27 ; Right edge 17 ; DQ delay 5 ; DQS delay 4
SEQ.C: Write Deskew ; DQ 15 ; Rank 0 ; Left edge 27 ; Right edge 17 ; DQ delay 6 ; DQS delay 4
SEQ.C: DM Deskew ; Group 0 ; Left edge 27 ; Right edge 18 ; DQ delay 4
SEQ.C: Read after Write ; DQ 8 ; Rank 0 ; Left edge 21 ; Right edge 20 ; DQ delay 1 ; DQS delay 12
SEQ.C: Read after Write ; DQ 9 ; Rank 0 ; Left edge 31 ; Right edge 9 ; DQ delay 11 ; DQS delay 12
SEQ.C: Read after Write ; DQ 10 ; Rank 0 ; Left edge 20 ; Right edge 20 ; DQ delay 11 ; DQS delay 12
SEQ.C: Read after Write ; DQ 11 ; Rank 0 ; Left edge 31 ; Right edge 9 ; DQ delay 12 ; DQS delay 12
SEQ.C: Read after Write ; DQ 12 ; Rank 0 ; Left edge 21 ; Right edge 1 ; DQ delay 1 ; DQS delay 12
```

### 7.14.3.1. Analysis of Debug Report

The following analysis will help you interpret the debug report.

- The Read Deskew and Write Deskew results shown in the debug report are before calibration. (Before calibration results are actually from the window seen *during* calibration, and are most useful for debugging.)
- For each DQ group, the Write Deskew, Read Deskew, DM Deskew, and Read after Write results map to the before-calibration margins reported in the EMIF Debug Toolkit.
- Note:** The Write Deskew, Read Deskew, DM Deskew, and Read after Write results are reported in delay steps (nominally 25ps, in Arria V and Cyclone V devices), not in picoseconds.
- DQS Enable calibration is reported as a VFIFO setting (in one clock period steps), a phase tap (in one-eighth clock period steps), and a delay chain step (in 25ps steps).

```
SEQ.C: DQS Enable ; Group 0 ; Rank 0 ; Start VFIFO 5 ; Phase 6 ; Delay 4
SEQ.C: DQS Enable ; Group 0 ; Rank 0 ; End VFIFO 6 ; Phase 5 ; Delay 9
SEQ.C: DQS Enable ; Group 0 ; Rank 0 ; Center VFIFO 6 ; Phase 2 ; Delay 1
```

Analysis of DQS Enable results: A VFIFO tap is 1 clock period, a phase is 1/8 clock period (45 degrees) and delay is nominally 25ps per tap. The DQS<sub>en</sub> window is the difference between the start and end—for the above example, assuming a frequency of 400 MHz (2500ps), that calculates as follows: start is  $5 * 2500 + 6 * 2500 / 8 + 4 * 25 = 14475$ ps. By the same calculation, the end is 16788ps. Consequently, the DQS<sub>en</sub> window is 2313ps.



- The size of a read window or write window is equal to (left edge + right edge) \* delay chain step size. Both the left edge and the right edge can be negative or positive.:

```
SEQ.C: Read Deskew ; DQ 0 ; Rank 0 ; Left edge 18 ; Right edge 27 ; DQ
delay 0 ; DQS delay 8
SEQ.C: Write Deskew ; DQ 0 ; Rank 0 ; Left edge 30 ; Right edge 17 ; DQ
delay 6 ; DQS delay 4
```

Analysis of DQ and DQS delay results: The DQ and DQS output delay (write) is the D5 delay chain. The DQ input delay (read) is the D1 delay chain, the DQS input delay (read) is the D4 delay chain.

- Consider the following example of latency results:

```
SEQ.C: LFIFO Calibration ; Latency 10
```

Analysis of latency results: This is the calibrated PHY read latency. The EMIF Debug Toolkit does not report this figure. This latency is reported in clock cycles.

- Consider the following example of FOM results:

```
SEQ.C: FOM IN = 83
SEQ.C: FOM OUT = 91
```

Analysis of FOM results: The FOM IN value is a measure of the health of the read interface; it is calculated as the sum over all groups of the minimum margin on DQ plus the margin on DQS, divided by 2. The FOM OUT is a measure of the health of the write interface; it is calculated as the sum over all groups of the minimum margin on DQ plus the margin on DQS, divided by 2. You may refer to these values as indicators of improvement when you are experimenting with various termination schemes, assuming there are no individual misbehaving DQ pins.

- The debug report does not provide delay chain step size values. The delay chain step size varies with device speed grade. Refer to your device data sheet for exact incremental delay values for delay chains.

### Related Information

#### Functional Description—UniPHY

For more information about calibration, refer to the *Calibration Stages* section in the *Functional Description—UniPHY* chapter of the *External Memory Interface Handbook*.

### 7.14.4. Writing a Predefined Data Pattern to SDRAM in the Preloader

You can include your own code to write a predefined data pattern to the SDRAM in the preloader for debugging purposes.

- Include your code in the file: <project\_folder>\software\spl\_bsp\uboot-socfpga\arch\arm\cpu\armv7\socfpga\spl.c .

Adding the following code to the spl.c file causes the controller to write walking 1s and walking 0s, repeated five times, to the SDRAM.

```
/*added for demo, place after the last #define statement in spl.c */
#define ROTATE_RIGHT(X) ( (X>>1) | (X&1?0X80000000:0) )
/*added for demo, place after the calibration code */
    test_data_walk0((long *)0x100000,PHYS_SDRAM_1_SIZE);
int test_data_walk0(long *base, long maxsize)
{
    volatile long *addr;
    long cnt;
```

```

ulong          data_temp[3];
ulong          expected_data[3];
ulong          read_data;
int           i = 0; //counter to loop different data pattern
int           num_address;

num_address=50;

data_temp[0]=0xFFFFFFFFFE; //initial data for walking 0 pattern
data_temp[1]=0X00000001; //initial data for walking 1 pattern
data_temp[2]=0XAAAAAAA; //initial data for A->5 switching

expected_data[0]=0xFFFFFFFFFE; //initial data for walking 0 pattern
expected_data[1]=0X00000001; //initial data for walking 1 pattern
expected_data[2]=0XAAAAAAA; //initial data for A->5
switching

for (i=0;i<3;i++) {

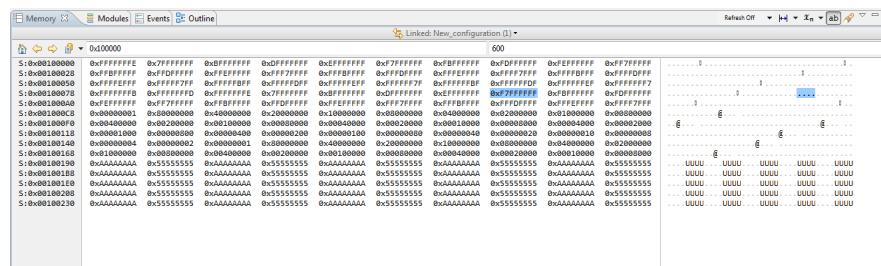
printf("\nSTARTED %08X DATA PATTERN !!!!\n",data_temp[i]);
/*write*/
for (cnt = (0+i*num_address); cnt < ((i+1)*num_address) ; cnt++ ) {
    addr = base + cnt; /* pointer arith! */
    sync ();
    *addr = data_temp[i];
    data_temp[i]=ROTATE_RIGHT(data_temp[i]);
}

/*read*/
for (cnt = (0+i*num_address); cnt < ((i+1)*num_address) ; cnt = cnt+
+ ) {
    addr = base + cnt; /* pointer arith! */
    sync ();
    read_data=*addr;
    printf("Address:%X Expected: %08X     Read:%08X \n",addr,
expected_data[i],read_data);
    if (expected_data[i] !=read_data) {
        puts("!!!!!!FAILED!!!!!!\n\n");
        hang();
    }
    expected_data[i]=ROTATE_RIGHT(expected_data[i]);
}
}
}
}

=====//End Of Code//=====

```

**Figure 155. Memory Contents After Executing Example Code**





## 7.15. SDRAM Controller Address Map and Register Definitions

This section lists the SDRAM register address map and describes the registers.

### 7.15.1. SDRAM Controller Address Map

Address map for the SDRAM Interface registers

Base Address: 0xFFC20000

#### SDRAM Controller Module

| Register                                    | Offset | Width | Access | Reset Value | Description                                  |
|---|--------|-------|--------|-------------|--|
| <a href="#">ctrlcfg</a> on page 309         | 0x5000 | 32    | RW     | 0x0         | Controller Configuration Register            |
| <a href="#">dramtiming1</a> on page 311     | 0x5004 | 32    | RW     | 0x0         | DRAM Timings 1 Register                      |
| <a href="#">dramtiming2</a> on page 312     | 0x5008 | 32    | RW     | 0x0         | DRAM Timings 2 Register                      |
| <a href="#">dramtiming3</a> on page 312     | 0x500C | 32    | RW     | 0x0         | DRAM Timings 3 Register                      |
| <a href="#">dramtiming4</a> on page 313     | 0x5010 | 32    | RW     | 0x0         | DRAM Timings 4 Register                      |
| <a href="#">lowpwrtiming</a> on page 314    | 0x5014 | 32    | RW     | 0x0         | Lower Power Timing Register                  |
| <a href="#">dramodt</a> on page 315         | 0x5018 | 32    | RW     | 0x0         | ODT Control Register                         |
| <a href="#">dramaddrw</a> on page 315       | 0x502C | 32    | RW     | 0x0         | DRAM Address Widths Register                 |
| <a href="#">dramifwidth</a> on page 316     | 0x5030 | 32    | RW     | 0x0         | DRAM Interface Data Width Register           |
| <a href="#">dramsts</a> on page 317         | 0x5038 | 32    | RW     | 0x0         | DRAM Status Register                         |
| <a href="#">dramintr</a> on page 318        | 0x503C | 32    | RW     | 0x0         | ECC Interrupt Register                       |
| <a href="#">sbecount</a> on page 318        | 0x5040 | 32    | RW     | 0x0         | ECC Single Bit Error Count Register          |
| <a href="#">dbeccount</a> on page 319       | 0x5044 | 32    | RW     | 0x0         | ECC Double Bit Error Count Register          |
| <a href="#">erraddr</a> on page 319         | 0x5048 | 32    | RW     | 0x0         | ECC Error Address Register                   |
| <a href="#">dropcount</a> on page 320       | 0x504C | 32    | RW     | 0x0         | ECC Auto-correction Dropped Count Register   |
| <a href="#">dropaddr</a> on page 320        | 0x5050 | 32    | RW     | 0x0         | ECC Auto-correction Dropped Address Register |
| <a href="#">lowpwreq</a> on page 321        | 0x5054 | 32    | RW     | 0x0         | Low Power Control Register                   |
| <a href="#">lowpwrack</a> on page 322       | 0x5058 | 32    | RW     | 0x0         | Low Power Acknowledge Register               |
| <a href="#">staticcfg</a> on page 323       | 0x505C | 32    | RW     | 0x0         | Static Configuration Register                |
| <a href="#">ctrlwidth</a> on page 323       | 0x5060 | 32    | RW     | 0x0         | Memory Controller Width Register             |
| <a href="#">portcfg</a> on page 324         | 0x507C | 32    | RW     | 0x0         | Port Configuration Register                  |
| <a href="#">fpgaportrst</a> on page 326     | 0x5080 | 32    | RW     | 0x0         | FPGA Ports Reset Control Register            |
| <a href="#">protportdefault</a> on page 327 | 0x508C | 32    | RW     | 0x0         | Memory Protection Port Default Register      |
| <a href="#">protruleaddr</a> on page 328    | 0x5090 | 32    | RW     | 0x0         | Memory Protection Address Register           |
| <a href="#">protruleid</a> on page 328      | 0x5094 | 32    | RW     | 0x0         | Memory Protection ID Register                |

*continued...*



| Register                                  | Offset | Width | Access | Reset Value | Description                                     |
|---|--------|-------|--------|-------------|---|
| <a href="#">protruledata</a> on page 329  | 0x5098 | 32    | RW     | 0x0         | Memory Protection Rule Data Register            |
| <a href="#">protrulerdwr</a> on page 330  | 0x509C | 32    | RW     | 0x0         | Memory Protection Rule Read-Write Register      |
| <a href="#">mppriority</a> on page 331    | 0x50AC | 32    | RW     | 0x0         | Scheduler priority Register                     |
| <a href="#">remappriority</a> on page 332 | 0x50E0 | 32    | RW     | 0x0         | Controller Command Pool Priority Remap Register |

### Port Sum of Weight Register

| Register                                 | Offset | Width | Access | Reset Value | Description                      |
|--|--------|-------|--------|-------------|----------------------------------|
| <a href="#">mpweight_0_4</a> on page 333 | 0x50B0 | 32    | RW     | 0x0         | Port Sum of Weight Register[1/4] |
| <a href="#">mpweight_1_4</a> on page 333 | 0x50B4 | 32    | RW     | 0x0         | Port Sum of Weight Register[2/4] |
| <a href="#">mpweight_2_4</a> on page 334 | 0x50B8 | 32    | RW     | 0x0         | Port Sum of Weight Register[3/4] |
| <a href="#">mpweight_3_4</a> on page 334 | 0x50BC | 32    | RW     | 0x0         | Port Sum of Weight Register[4/4] |

### 7.15.1.1. SDRAM Controller Module Register Descriptions

Address map for the SDRAM controller and multi-port front-end. All registers in this group reset to zero.

Offset: 0x5000

- [ctrlcfg](#) on page 309
- [dramtiming1](#) on page 311
- [dramtiming2](#) on page 312
- [dramtiming3](#) on page 312
- [dramtiming4](#) on page 313
- [lowpwrtiming](#) on page 314
- [dramodt](#) on page 315
- [dramaddrw](#) on page 315
- [dramifwidth](#) on page 316
- [dramsts](#) on page 317
- [dramintr](#) on page 318
- [sbecount](#) on page 318
- [dbecount](#) on page 319
- [erraddr](#) on page 319
- [dropcount](#) on page 320
- [dropaddr](#) on page 320
- [lowpwreq](#) on page 321
- [lowpwrack](#) on page 322
- [staticcfg](#) on page 323



[ctrlwidth](#) on page 323  
[portcfg](#) on page 324  
[fpgaportrst](#) on page 326  
[protportdefault](#) on page 327  
[proruleaddr](#) on page 328  
[proruleid](#) on page 328  
[proruledata](#) on page 329  
[protrulerdwr](#) on page 330  
[mppriority](#) on page 331  
[remappriority](#) on page 332  
[Port Sum of Weight Register Register Descriptions](#) on page 333

#### 7.15.1.1.1. ctrlcfg

The Controller Configuration Register determines the behavior of the controller.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25000       |

Offset: 0x5000

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields                 |                     |                     |   |                           |                    |                     |                              |                              |                           |                           |                       |    |                   |    |    |  |  |
|----------------------------|---------------------|---------------------|---|---------------------------|--------------------|---------------------|------------------------------|------------------------------|---------------------------|---------------------------|-----------------------|----|-------------------|----|----|--|--|
| 31                         | 30                  | 29                  | 28  | 27                        | 26                 | 25                  | 24                           | 23                           | 22                        | 21                        | 20                    | 19 | 18                | 17 | 16 |  |  |
| Reserved                   |                     |                     |   |                           |                    |                     | burst<br>termen<br>RW<br>0x0 | burst<br>intren<br>RW<br>0x0 | nodmp<br>ins<br>RW<br>0x0 | dqstr<br>ken<br>RW<br>0x0 | starvelimit<br>RW 0x0 |    |                   |    |    |  |  |
| 15                         | 14                  | 13                  | 12  | 11                        | 10                 | 9                   | 8                            | 7                            | 6                         | 5                         | 4                     | 3  | 2                 | 1  | 0  |  |  |
| reord<br>eren<br>RW<br>0x0 | gendbe<br>RW<br>0x0 | gensbe<br>RW<br>0x0 | cfg_enable<br>_ecc_code_overwrites<br>RW<br>0x0 | ecco<br>rren<br>RW<br>0x0 | eccen<br>RW<br>0x0 | addrorder<br>RW 0x0 |                              | membl<br>RW 0x0              |                           |                           |                       |    | memtype<br>RW 0x0 |    |    |  |  |

**ctrlcfg Fields**

| Bit       | Name                           | Description  | Access | Reset       |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
|-----------|--------------------------------|--|--------|-------------|----------------------|-----|-------------------------|--|-----|-------------------------|-----------------|-----|-------------------------|---|-----|----------|-----|----|-----|
| 25        | burstterminen                  | Set to a one to enable the controller to issue burst terminate commands. This must only be set when the DRAM memory type is LPDDR2.  | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 24        | burstintren                    | Set to a one to enable the controller to issue burst interrupt commands. This must only be set when the DRAM memory type is LPDDR2.  | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 23        | nodmpins                       | Set to a one to enable DRAM operation if no DM pins are connected.   | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 22        | dqstrken                       | Enables DQS tracking in the PHY.   | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 21:1<br>6 | starvelimit                    | Specifies the number of DRAM burst transactions an individual transaction will allow to reorder ahead of it before its priority is raised in the memory controller.  | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 15        | reorderen                      | This bit controls whether the controller can re-order operations to optimize SDRAM bandwidth. It should generally be set to a one.   | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 14        | gendbe                         | Enable the deliberate insertion of double bit errors in data written to memory. This should only be used for testing purposes.   | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 13        | gensbe                         | Enable the deliberate insertion of single bit errors in data written to memory. This should only be used for testing purposes.   | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 12        | cfg_enable_ecc_code_overwrites | Set to a one to enable ECC overwrites. ECC overwrites occur when a correctable ECC error is seen and cause a new read/modify/write to be scheduled for that location to clear the ECC error.   | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 11        | ecccorgen                      | Enable auto correction of the read data returned when single bit error is detected.  | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 10        | eccen                          | Enable the generation and checking of ECC. This bit must only be set if the memory connected to the SDRAM interface is 24 or 40 bits wide. If you set this, you must clear the useeccasdata field in the staticcfg register.   | RW     | 0x0         |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 9:8       | addrorder                      | This bit field selects the order for address interleaving. Programming this field with different values gives different mappings between the AXI or Avalon-MM address and the SDRAM address. Program this field with the following binary values to select the ordering.<br><br><table border="1"><thead><tr><th>Value</th><th>Description</th><th>Address Interleaving</th></tr></thead><tbody><tr><td>0x0</td><td>chip, row, bank, column</td><td>Bank interleaved with no rank (chip select) interleaving</td></tr><tr><td>0x1</td><td>chip, bank, row, column</td><td>No interleaving</td></tr><tr><td>0x2</td><td>row, chip, bank, column</td><td>Bank interleaved with rank (chip select) interleaving</td></tr><tr><td>0x3</td><td>reserved</td><td>N/A</td></tr></tbody></table> | Value  | Description | Address Interleaving | 0x0 | chip, row, bank, column | Bank interleaved with no rank (chip select) interleaving | 0x1 | chip, bank, row, column | No interleaving | 0x2 | row, chip, bank, column | Bank interleaved with rank (chip select) interleaving | 0x3 | reserved | N/A | RW | 0x0 |
| Value     | Description                    | Address Interleaving   |        |             |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 0x0       | chip, row, bank, column        | Bank interleaved with no rank (chip select) interleaving   |        |             |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 0x1       | chip, bank, row, column        | No interleaving  |        |             |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 0x2       | row, chip, bank, column        | Bank interleaved with rank (chip select) interleaving  |        |             |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |
| 0x3       | reserved                       | N/A  |        |             |                      |     |                         |  |     |                         |                 |     |                         |   |     |          |     |    |     |

*continued...*



| Bit     | Name                        | Description   | Access | Reset       |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
|---------|-----------------------------|---|--------|-------------|-----|----------|-----|---------------------------|-----|---------------------------|-----|----------|-----|-----------------------------|---------|----------|----|-----|
|         |                             | Intel recommends programming addrorder to 0x0 or 0x2.   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 7:3     | membl                       | Configures burst length as a static decimal value. Legal values are valid for JEDEC allowed DRAM values for the DRAM selected in cfg_type. For DDR3, this should be programmed with 8 (binary "01000"), for DDR2 it can be either 4 or 8 depending on the exact DRAM chip. LPDDR2 can be programmed with 4, 8, or 16 and LPDDR can be programmed with 2, 4, or 8. You must also program the membl field in the staticcfg register.  | RW     | 0x0         |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 2:0     | memtype                     | <p>This bit field selects the memory type. This field can be programmed with the following binary values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0x0</td><td>Reserved</td></tr> <tr> <td>0x1</td><td>Memory type is DDR2 SDRAM</td></tr> <tr> <td>0x2</td><td>Memory type is DDR3 SDRAM</td></tr> <tr> <td>0x3</td><td>reserved</td></tr> <tr> <td>0x4</td><td>Memory type is LPDDR2 SDRAM</td></tr> <tr> <td>0x5-0x7</td><td>Reserved</td></tr> </tbody> </table> | Value  | Description | 0x0 | Reserved | 0x1 | Memory type is DDR2 SDRAM | 0x2 | Memory type is DDR3 SDRAM | 0x3 | reserved | 0x4 | Memory type is LPDDR2 SDRAM | 0x5-0x7 | Reserved | RW | 0x0 |
| Value   | Description                 |   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 0x0     | Reserved                    |   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 0x1     | Memory type is DDR2 SDRAM   |   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 0x2     | Memory type is DDR3 SDRAM   |   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 0x3     | reserved                    |   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 0x4     | Memory type is LPDDR2 SDRAM |   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |
| 0x5-0x7 | Reserved                    |   |        |             |     |          |     |                           |     |                           |     |          |     |                             |         |          |    |     |

### 7.15.1.1.2. dramtiming1

This register implements JEDEC standardized timing parameters. It should be programmed in clock cycles, for the value specified by the memory vendor.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25004       |

Offset: 0x5004

Access: RW

| Bit Fields     |               |    |    |    |    |    |               |    |                |    |    |    |                |    |                |  |  |
|----------------|---------------|----|----|----|----|----|---------------|----|----------------|----|----|----|----------------|----|----------------|--|--|
| 31             | 30            | 29 | 28 | 27 | 26 | 25 | 24            | 23 | 22             | 21 | 20 | 19 | 18             | 17 | 16             |  |  |
| trfc<br>RW 0x0 |               |    |    |    |    |    |               |    | tfaw<br>RW 0x0 |    |    |    |                |    | trrd<br>RW 0x0 |  |  |
| 15             | 14            | 13 | 12 | 11 | 10 | 9  | 8             | 7  | 6              | 5  | 4  | 3  | 2              | 1  | 0              |  |  |
| trrd<br>RW 0x0 | tcl<br>RW 0x0 |    |    |    |    |    | tal<br>RW 0x0 |    |                |    |    |    | tcwl<br>RW 0x0 |    |                |  |  |

#### dramtiming1 Fields

| Bit                 | Name | Description                                | Access | Reset |
|---------------------|------|--|--------|-------|
| 31:2<br>4           | trfc | The refresh cycle timing parameter.        | RW     | 0x0   |
| 23:1<br>8           | tfaw | The four-activate window timing parameter. | RW     | 0x0   |
| <i>continued...</i> |      |  |        |       |



| Bit       | Name | Description   | Access | Reset |
|-----------|------|---|--------|-------|
| 17:1<br>4 | trrd | The activate to activate, different banks timing parameter. | RW     | 0x0   |
| 13:9      | tcl  | Memory read latency.  | RW     | 0x0   |
| 8:4       | tal  | Memory additive latency.                                    | RW     | 0x0   |
| 3:0       | tcwl | Memory write latency.                                       | RW     | 0x0   |

### 7.15.1.1.3. dramtiming2

This register implements JEDEC standardized timing parameters. It should be programmed in clock cycles, for the value specified by the memory vendor.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25008       |

Offset: 0x5008

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields     |    |    |                 |    |    |    |               |    |    |    |               |    |    |    |                |
|----------------|----|----|-----------------|----|----|----|---------------|----|----|----|---------------|----|----|----|----------------|
| 31             | 30 | 29 | 28              | 27 | 26 | 25 | 24            | 23 | 22 | 21 | 20            | 19 | 18 | 17 | 16             |
| Reserved       |    |    | twtr<br>RW 0x0  |    |    |    | tvr<br>RW 0x0 |    |    |    | trp<br>RW 0x0 |    |    |    | trcd<br>RW 0x0 |
| 15             | 14 | 13 | 12              | 11 | 10 | 9  | 8             | 7  | 6  | 5  | 4             | 3  | 2  | 1  | 0              |
| trcd<br>RW 0x0 |    |    | trefi<br>RW 0x0 |    |    |    |               |    |    |    |               |    |    |    |                |

### dramtiming2 Fields

| Bit       | Name  | Description                                  | Access | Reset |
|-----------|-------|--|--------|-------|
| 28:2<br>5 | twtr  | The write to read timing parameter.          | RW     | 0x0   |
| 24:2<br>1 | tvr   | The write recovery timing.                   | RW     | 0x0   |
| 20:1<br>7 | trp   | The precharge to activate timing parameter.  | RW     | 0x0   |
| 16:1<br>3 | trcd  | The activate to read/write timing parameter. | RW     | 0x0   |
| 12:0      | trefi | The refresh interval timing parameter.       | RW     | 0x0   |

### 7.15.1.1.4. dramtiming3

This register implements JEDEC standardized timing parameters. It should be programmed in clock cycles, for the value specified by the memory vendor.



| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2500C       |

Offset: 0x500C

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields     |               |    |    |    |    |    |    |                |                 |    |    |    |                |    |                |  |
|----------------|---------------|----|----|----|----|----|----|----------------|-----------------|----|----|----|----------------|----|----------------|--|
| 31             | 30            | 29 | 28 | 27 | 26 | 25 | 24 | 23             | 22              | 21 | 20 | 19 | 18             | 17 | 16             |  |
| Reserved       |               |    |    |    |    |    |    |                | tcccd<br>RW 0x0 |    |    |    | tmrd<br>RW 0x0 |    |                |  |
| 15             | 14            | 13 | 12 | 11 | 10 | 9  | 8  | 7              | 6               | 5  | 4  | 3  | 2              | 1  | 0              |  |
| tmrd<br>RW 0x0 | trc<br>RW 0x0 |    |    |    |    |    |    | tras<br>RW 0x0 |                 |    |    |    |                |    | trtp<br>RW 0x0 |  |

### dramtiming3 Fields

| Bit       | Name  | Description                                 | Access | Reset |
|-----------|-------|---|--------|-------|
| 22:1<br>9 | tcccd | The CAS to CAS delay time.                  | RW     | 0x0   |
| 18:1<br>5 | tmrd  | Mode register timing parameter.             | RW     | 0x0   |
| 14:9      | trc   | The activate to activate timing parameter.  | RW     | 0x0   |
| 8:4       | tras  | The activate to precharge timing parameter. | RW     | 0x0   |
| 3:0       | trtp  | The read to precharge timing parameter.     | RW     | 0x0   |

### 7.15.1.1.5. dramtiming4

This register implements JEDEC standardized timing parameters. It should be programmed in clock cycles, for the value specified by the memory vendor.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25010       |

Offset: 0x5010

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.



| Bit Fields            |    |    |    |    |    |    |    |                            |    |    |    |                       |    |    |    |  |
|-----------------------|----|----|----|----|----|----|----|----------------------------|----|----|----|-----------------------|----|----|----|--|
| 31                    | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23                         | 22 | 21 | 20 | 19                    | 18 | 17 | 16 |  |
| Reserved              |    |    |    |    |    |    |    | minpwrsavecycles<br>RW 0x0 |    |    |    | pwrdownexit<br>RW 0x0 |    |    |    |  |
| 15                    | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7                          | 6  | 5  | 4  | 3                     | 2  | 1  | 0  |  |
| pwrdownexit<br>RW 0x0 |    |    |    |    |    |    |    | selfrfshexit<br>RW 0x0     |    |    |    |                       |    |    |    |  |

#### dramtiming4 Fields

| Bit       | Name             | Description   | Access | Reset |
|-----------|------------------|---|--------|-------|
| 23:2<br>0 | minpwrsavecycles | The minimum number of cycles to stay in a low power state. This applies to both power down and self-refresh and should be set to the greater of tPD and tCKESR. | RW     | 0x0   |
| 19:1<br>0 | pwrdownexit      | The power down exit cycles, tXPDLL.   | RW     | 0x0   |
| 9:0       | selfrfshexit     | The self refresh exit cycles, tXS.  | RW     | 0x0   |

#### 7.15.1.1.6. lowpwrtiming

This register controls the behavior of the low power logic in the controller.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25014       |

Offset: 0x5014

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields             |    |    |    |    |    |    |    |                            |    |    |    |    |    |    |    |  |
|------------------------|----|----|----|----|----|----|----|----------------------------|----|----|----|----|----|----|----|--|
| 31                     | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23                         | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| Reserved               |    |    |    |    |    |    |    | clkdisablecycles<br>RW 0x0 |    |    |    |    |    |    |    |  |
| 15                     | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7                          | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| autopdcycles<br>RW 0x0 |    |    |    |    |    |    |    |                            |    |    |    |    |    |    |    |  |

#### lowpwrtiming Fields

| Bit       | Name             | Description  | Access | Reset |
|-----------|------------------|--|--------|-------|
| 19:1<br>6 | clkdisablecycles | Set to the number of clocks after the execution of a self-refresh to stop the clock. This register is generally set based on PHY design latency and should generally not be changed. | RW     | 0x0   |
| 15:0      | autopdcycles     | The number of idle clock cycles after which the controller should place the memory into power-down mode.   | RW     | 0x0   |



### 7.15.1.1.7. dramodt

This register controls which ODT pin asserts with chip select 0 (CS0) assertion and which ODT pin asserts with chip select 1 (CS1) assertion.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25018       |

Offset: 0x5018

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |                             |    |    |    |                              |    |  |
|------------|----|----|----|----|----|----|----|----|----|-----------------------------|----|----|----|------------------------------|----|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21                          | 20 | 19 | 18 | 17                           | 16 |  |
| Reserved   |    |    |    |    |    |    |    |    |    |                             |    |    |    |                              |    |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5                           | 4  | 3  | 2  | 1                            | 0  |  |
| Reserved   |    |    |    |    |    |    |    |    |    | cfg_read_odt_chip<br>RW 0x0 |    |    |    | cfg_write_odt_chip<br>RW 0x0 |    |  |

#### dramodt Fields

| Bit | Name               | Description   | Access | Reset |
|-----|--------------------|---|--------|-------|
| 7:4 | cfg_read_odt_chip  | This register controls which ODT pin is asserted during reads. Bits[5:4] select the ODT pin that asserts with CS0 and bits[7:6] select the ODT pin that asserts with CS1. For example, a value of 0x9 asserts ODT[0] for accesses CS0 and ODT[1] for accesses with CS1. This field can be set to 0x1 if there is only one chip select available.  | RW     | 0x0   |
| 3:0 | cfg_write_odt_chip | This register controls which ODT pin is asserted during writes. Bits[1:0] select the ODT pin that asserts with CS0 and bits[3:2] select the ODT pin that asserts with CS1. For example, a value of 0x9 asserts ODT[0] for accesses CS0 and ODT[1] for accesses with CS1. This field can be set to 0x1 if there is only one chip select available. | RW     | 0x0   |

### 7.15.1.1.8. dramaddrw

This register configures the width of the various address fields of the DRAM. The values specified in this register must match the memory devices being used.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2502C       |

Offset: 0x502C

Access: RW



**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields       |    |    |                    |    |    |                   |    |    |    |    |                   |    |    |    |    |  |
|------------------|----|----|--------------------|----|----|-------------------|----|----|----|----|-------------------|----|----|----|----|--|
| 31               | 30 | 29 | 28                 | 27 | 26 | 25                | 24 | 23 | 22 | 21 | 20                | 19 | 18 | 17 | 16 |  |
| Reserved         |    |    |                    |    |    |                   |    |    |    |    |                   |    |    |    |    |  |
| 15               | 14 | 13 | 12                 | 11 | 10 | 9                 | 8  | 7  | 6  | 5  | 4                 | 3  | 2  | 1  | 0  |  |
| csbits<br>RW 0x0 |    |    | bankbits<br>RW 0x0 |    |    | rowbits<br>RW 0x0 |    |    |    |    | colbits<br>RW 0x0 |    |    |    |    |  |

### dramaddrw Fields

| Bit       | Name     | Description   | Access | Reset |
|-----------|----------|---|--------|-------|
| 15:1<br>3 | csbits   | This field defines the number of chip select address bits. Set this field to 0x0 for single chip select and to 0x1 for two chip selects.<br><br>When this field is set to 0x1, you may use rank interleaved mode by programming the ctrcfg.addorder field to 0x2. If you are using a single rank memory interface (csbits=0x0), you may not enable the rank interleaved mode (ctrcfg.addorder must be set less than 0x2).<br><br>When this field is set to 0x1 to enable dual ranks, the chip select (cs) bit of the incoming address is used to determine which chip select is active. When the chip select bit of the incoming address is 0, chip select 0 becomes active. When the chip select bit of the incoming address is 1, chip select 1 becomes active. | RW     | 0x0   |
| 12:1<br>0 | bankbits | The number of bank address bits for the memory devices in your memory interface.  | RW     | 0x0   |
| 9:5       | rowbits  | The number of row address bits for the memory devices in your memory interface.   | RW     | 0x0   |
| 4:0       | colbits  | The number of column address bits for the memory devices in your memory interface.  | RW     | 0x0   |

#### 7.15.1.1.9. dramifwidth

This register controls the interface width of the SDRAM controller.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25030       |

Offset: 0x5030

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.



| Bit Fields |    |    |    |    |    |    |    |    |    |                   |    |    |    |    |    |  |
|------------|----|----|----|----|----|----|----|----|----|-------------------|----|----|----|----|----|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21                | 20 | 19 | 18 | 17 | 16 |  |
| Reserved   |    |    |    |    |    |    |    |    |    |                   |    |    |    |    |    |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5                 | 4  | 3  | 2  | 1  | 0  |  |
| Reserved   |    |    |    |    |    |    |    |    |    | ifwidth<br>RW 0x0 |    |    |    |    |    |  |

### dramifwidth Fields

| Bit | Name    | Description  | Access | Reset |
|-----|---------|--|--------|-------|
| 7:0 | ifwidth | This register controls the width of the SDRAM interface, including any bits used for ECC. For example, for a 32-bit interface with ECC, program this register to 0x28. The ctrlwidth register must also be programmed. | RW     | 0x0   |

### 7.15.1.1.10. dramsts

This register provides the status of the calibration and ECC logic.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25038       |

Offset: 0x5038

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |                              |                          |                         |                            |                                 |    |  |  |
|------------|----|----|----|----|----|----|----|----|----|------------------------------|--------------------------|-------------------------|----------------------------|---------------------------------|----|--|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21                           | 20                       | 19                      | 18                         | 17                              | 16 |  |  |
| Reserved   |    |    |    |    |    |    |    |    |    |                              |                          |                         |                            |                                 |    |  |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5                            | 4                        | 3                       | 2                          | 1                               | 0  |  |  |
| Reserved   |    |    |    |    |    |    |    |    |    | corrdrop<br>rop<br>RW<br>0x0 | dbeerr<br>r<br>RW<br>0x0 | sbeer<br>r<br>RW<br>0x0 | calfail<br>il<br>RW<br>0x0 | calsuccess<br>cess<br>RW<br>0x0 |    |  |  |

### dramsts Fields

| Bit | Name       | Description  | Access | Reset |
|-----|------------|--|--------|-------|
| 4   | corrdrop   | This bit is set to 1 when any auto-corrections have been dropped.        | RW     | 0x0   |
| 3   | dbeerr     | This bit is set to 1 when any ECC double bit errors are detected.        | RW     | 0x0   |
| 2   | sbeer      | This bit is set to 1 when any ECC single bit errors are detected.        | RW     | 0x0   |
| 1   | calfail    | This bit is set to 1 when the PHY is unable to calibrate.                | RW     | 0x0   |
| 0   | calsuccess | This bit will be set to 1 if the PHY was able to successfully calibrate. | RW     | 0x0   |



### 7.15.1.1.11. dramintr

This register can enable, disable and clear the SDRAM error interrupts.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2503C       |

Offset: 0x503C

Access: RW

*Important:* To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |    |                          |                                   |                          |                          |                          |  |
|------------|----|----|----|----|----|----|----|----|----|----|--------------------------|-----------------------------------|--------------------------|--------------------------|--------------------------|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20                       | 19                                | 18                       | 17                       | 16                       |  |
| Reserved   |    |    |    |    |    |    |    |    |    |    |                          |                                   |                          |                          |                          |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4                        | 3                                 | 2                        | 1                        | 0                        |  |
| Reserved   |    |    |    |    |    |    |    |    |    |    |                          |                                   |                          |                          |                          |  |
|            |    |    |    |    |    |    |    |    |    |    | intrc<br>lr<br>RW<br>0x0 | corrd<br>ropma<br>sk<br>RW<br>0x0 | dbema<br>sk<br>RW<br>0x0 | sbema<br>sk<br>RW<br>0x0 | intren<br>n<br>RW<br>0x0 |  |

#### dramintr Fields

| Bit | Name         | Description   | Access | Reset |
|-----|--------------|---|--------|-------|
| 4   | intrclr      | Writing to this self-clearing bit clears the interrupt signal. Writing to this bit also clears the error count and error address registers: sbecount, dbecount, dropcount, erraddr, and dropaddr. | RW     | 0x0   |
| 3   | corrdropmask | Set this bit to a one to mask interrupts for an ECC correction write back needing to be dropped. This indicates a burst of memory errors in a short period of time.                               | RW     | 0x0   |
| 2   | dbemask      | Mask the double bit error interrupt.  | RW     | 0x0   |
| 1   | sbemask      | Mask the single bit error interrupt.  | RW     | 0x0   |
| 0   | intren       | Enable the interrupt output.  | RW     | 0x0   |

### 7.15.1.1.12. sbecount

This register tracks the single-bit error count.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25040       |

Offset: 0x5040

Access: RW



**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |                 |    |    |    |    |    |  |
|------------|----|----|----|----|----|----|----|----|----|-----------------|----|----|----|----|----|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21              | 20 | 19 | 18 | 17 | 16 |  |
| Reserved   |    |    |    |    |    |    |    |    |    |                 |    |    |    |    |    |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5               | 4  | 3  | 2  | 1  | 0  |  |
| Reserved   |    |    |    |    |    |    |    |    |    | count<br>RW 0x0 |    |    |    |    |    |  |

### sbeccount Fields

| Bit | Name  | Description  | Access | Reset |
|-----|-------|--|--------|-------|
| 7:0 | count | Reports the number of single bit errors that have occurred since the status register counters were last cleared. | RW     | 0x0   |

### 7.15.1.1.13. dbecount

This register tracks the double-bit error count.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25044       |

Offset: 0x5044

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |                 |    |    |    |    |    |  |
|------------|----|----|----|----|----|----|----|----|----|-----------------|----|----|----|----|----|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21              | 20 | 19 | 18 | 17 | 16 |  |
| Reserved   |    |    |    |    |    |    |    |    |    |                 |    |    |    |    |    |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5               | 4  | 3  | 2  | 1  | 0  |  |
| Reserved   |    |    |    |    |    |    |    |    |    | count<br>RW 0x0 |    |    |    |    |    |  |

### dbecount Fields

| Bit | Name  | Description  | Access | Reset |
|-----|-------|--|--------|-------|
| 7:0 | count | Reports the number of double bit errors that have occurred since the status register counters were last cleared. | RW     | 0x0   |

### 7.15.1.1.14. erraddr

This register holds the address of the most recent ECC error.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25048       |



Offset: 0x5048

Access: RW

| Bit Fields     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 31             | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| addr<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 15             | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| addr<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |

#### erraddr Fields

| Bit  | Name | Description  | Access | Reset |
|------|------|--|--------|-------|
| 31:0 | addr | The address of the most recent ECC error.<br><i>Note:</i> For a 32-bit interface, ECC is calculated across a span of 8 bytes, meaning the error address is a multiple of 8 bytes (4 bytes*2 burst length). To find the byte address of the word that contains the error, you must multiply the value in the erraddr register by 8. | RW     | 0x0   |

#### 7.15.1.1.15. dropcount

This register holds the address of the most recent ECC error.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2504C       |

Offset: 0x504C

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |                         |    |    |    |    |    |  |
|------------|----|----|----|----|----|----|----|----|----|-------------------------|----|----|----|----|----|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21                      | 20 | 19 | 18 | 17 | 16 |  |
| Reserved   |    |    |    |    |    |    |    |    |    |                         |    |    |    |    |    |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5                       | 4  | 3  | 2  | 1  | 0  |  |
| Reserved   |    |    |    |    |    |    |    |    |    | corrdropcount<br>RW 0x0 |    |    |    |    |    |  |

#### dropcount Fields

| Bit | Name          | Description   | Access | Reset |
|-----|---------------|---|--------|-------|
| 7:0 | corrdropcount | This gives the count of the number of ECC write back transactions dropped due to the internal FIFO overflowing. | RW     | 0x0   |

#### 7.15.1.1.16. dropaddr

This register holds the last dropped address.



| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25050       |

Offset: 0x5050

Access: RW

| Bit Fields   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| 31    30    29    28    27    26    25    24    23    22    21    20    19    18    17    16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| corrdropaddr<br>RW 0x0   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15    14    13    12    11    10    9    8    7    6    5    4    3    2    1    0           |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| corrdropaddr<br>RW 0x0   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

### dropaddr Fields

| Bit  | Name         | Description   | Access | Reset |
|------|--------------|---|--------|-------|
| 31:0 | corrdropaddr | This register gives the last address which was dropped. | RW     | 0x0   |

### 7.15.1.1.17. lowpwreq

This register instructs the controller to put the DRAM into a power down state. Note that some commands are only valid for certain memory types.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25054       |

Offset: 0x5054

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields   |  |  |  |  |  |  |  |  |  |                        |                             |                             |                                   |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|------------------------|-----------------------------|-----------------------------|-----------------------------------|--|--|--|--|
| 31    30    29    28    27    26    25    24    23    22    21    20    19    18    17    16 |  |  |  |  |  |  |  |  |  |                        |                             |                             |                                   |  |  |  |  |
| Reserved   |  |  |  |  |  |  |  |  |  |                        |                             |                             |                                   |  |  |  |  |
| 15    14    13    12    11    10    9    8    7    6    5    4    3    2    1    0           |  |  |  |  |  |  |  |  |  |                        |                             |                             |                                   |  |  |  |  |
| Reserved   |  |  |  |  |  |  |  |  |  | selfrfshmask<br>RW 0x0 | selfr<br>shreq<br>RW<br>0x0 | deeppwrndmas<br>k<br>RW 0x0 | deepp<br>wrdrn<br>eq<br>RW<br>0x0 |  |  |  |  |



### lowpwreq Fields

| Bit | Name           | Description   | Access | Reset |
|-----|----------------|---|--------|-------|
| 5:4 | selfrfshmask   | Write a one to each bit of this field to have a self refresh request apply to both chips.   | RW     | 0x0   |
| 3   | selfrshreq     | Write a one to this bit to request the RAM be put into a self refresh state. This bit is treated as a static value so the RAM will remain in self-refresh as long as this register bit is set to a one. This power down mode can be selected for all DRAMs supported by the controller. | RW     | 0x0   |
| 2:1 | deeppwrdrnmask | Write ones to this register to select which DRAM chip selects will be powered down. Typical usage is to set both of these bits when deeppwrdrnreq is set but the controller does support putting a single chip into deep power down and keeping the other chip running.                 | RW     | 0x0   |
| 0   | deeppwrdrnreq  | Write a one to this bit to request a deep power down. This bit should only be written with LPDDR2 DRAMs, DDR3 DRAMs do not support deep power down.   | RW     | 0x0   |

#### 7.15.1.1.18. lowpwrack

This register gives the status of the power down commands requested by the Low Power Control register.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25058       |

Offset: 0x5058

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                          |                            |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------------------|----------------------------|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16                       |                            |
| Reserved   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                          |                            |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0                        |                            |
| Reserved   |    |    |    |    |    |    |    |    |    |    |    |    |    |    | selfrfshack<br>RW<br>0x0 | deeppwrdrnreq<br>RW<br>0x0 |

### lowpwrack Fields

| Bit | Name          | Description   | Access | Reset |
|-----|---------------|---|--------|-------|
| 1   | selfrfshack   | This bit is a one to indicate that the controller is in a self-refresh state. | RW     | 0x0   |
| 0   | deeppwrdrnreq | This bit is set to a one after a deep power down has been executed            | RW     | 0x0   |



### 7.15.1.1.19. staticcfg

This register controls configuration values which cannot be updated during active transfers. First configure the `membl` and `eccn` fields and then re-write these fields while setting the `applycfg` bit. The `applycfg` bit is write only.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2505C       |

Offset: 0x505C

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |    |    |                       |                           |                 |    |
|------------|----|----|----|----|----|----|----|----|----|----|----|-----------------------|---------------------------|-----------------|----|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19                    | 18                        | 17              | 16 |
| Reserved   |    |    |    |    |    |    |    |    |    |    |    |                       |                           |                 |    |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3                     | 2                         | 1               | 0  |
| Reserved   |    |    |    |    |    |    |    |    |    |    |    | applycfg<br>RW<br>0x0 | useeccasdata<br>RW<br>0x0 | membl<br>RW 0x0 |    |

#### staticcfg Fields

| Bit | Name         | Description   | Access | Reset |
|-----|--------------|---|--------|-------|
| 3   | applycfg     | Write with this bit set to apply all the settings loaded in SDR registers to the memory interface. This bit is write-only and always returns 0 if read.   | RW     | 0x0   |
| 2   | useeccasdata | This field allows the FPGA ports to directly access the extra data bits that are normally used to hold the ECC code. The interface width must be set to 24 or 40 in the <code>dramifwidth</code> register. If you set this, you must clear the <code>eccn</code> field in the <code>ctrlcfg</code> register.  | RW     | 0x0   |
| 1:0 | membl        | This field specifies the DRAM burst length. The following encodings set the burst length: <ul style="list-style-type: none"> <li>• 0x0= Burst length of 2 clocks</li> <li>• 0x1= Burst length of 4 clocks</li> <li>• 0x2= Burst length of 8 clocks</li> <li>• 0x3= Burst length of 16 clocks</li> </ul> If you program the this field, you must also set the <code>membl</code> field in the <code>ctrlcfg</code> register. | RW     | 0x0   |

### 7.15.1.1.20. ctrlwidth

This register controls the width of the physical DRAM interface.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25060       |



Offset: 0x5060

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                     |  |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------------------|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16                  |  |
| Reserved   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                     |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0                   |  |
| Reserved   |    |    |    |    |    |    |    |    |    |    |    |    |    |    | ctrlwidth<br>RW 0x0 |  |

#### ctrlwidth Fields

| Bit   | Name  | Description  | Access | Reset       |     |                       |     |   |     |   |    |     |
|-------|---|--|--------|-------------|-----|-----------------------|-----|---|-----|---|----|-----|
| 1:0   | ctrlwidth   | <p>This field specifies the SDRAM controller interface width:</p> <table border="1"><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>0x0</td><td>8-bit interface width</td></tr><tr><td>0x1</td><td>16-bit (no ECC) or 24-bit (ECC enabled) interface width</td></tr><tr><td>0x2</td><td>32-bit (no ECC) or 40-bit (ECC enabled) interface width</td></tr></tbody></table> <p>Additionally, you must program the dramifwidth register.</p> | Value  | Description | 0x0 | 8-bit interface width | 0x1 | 16-bit (no ECC) or 24-bit (ECC enabled) interface width | 0x2 | 32-bit (no ECC) or 40-bit (ECC enabled) interface width | RW | 0x0 |
| Value | Description   |  |        |             |     |                       |     |   |     |   |    |     |
| 0x0   | 8-bit interface width                                   |  |        |             |     |                       |     |   |     |   |    |     |
| 0x1   | 16-bit (no ECC) or 24-bit (ECC enabled) interface width |  |        |             |     |                       |     |   |     |   |    |     |
| 0x2   | 32-bit (no ECC) or 40-bit (ECC enabled) interface width |  |        |             |     |                       |     |   |     |   |    |     |

#### 7.15.1.1.21. portcfg

Each bit of the autopchen field maps to one of the control ports. If a port executes mostly sequential memory accesses, the corresponding autopchen bit should be 0. If the port has highly random accesses, then its autopchen bit should be set to 1.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2507C       |

Offset: 0x507C

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.



| Bit Fields          |    |    |    |    |    |          |    |    |    |                        |    |                     |    |    |    |  |
|---------------------|----|----|----|----|----|----------|----|----|----|------------------------|----|---------------------|----|----|----|--|
| 31                  | 30 | 29 | 28 | 27 | 26 | 25       | 24 | 23 | 22 | 21                     | 20 | 19                  | 18 | 17 | 16 |  |
| Reserved            |    |    |    |    |    |          |    |    |    |                        |    | autopchen<br>RW 0x0 |    |    |    |  |
| 15                  | 14 | 13 | 12 | 11 | 10 | 9        | 8  | 7  | 6  | 5                      | 4  | 3                   | 2  | 1  | 0  |  |
| autopchen<br>RW 0x0 |    |    |    |    |    | Reserved |    |    |    | portprotocol<br>RO 0x0 |    |                     |    |    |    |  |

### portcfg Fields

| Bit   | Name   | Description  | Access | Reset                |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
|-------|--|--|--------|----------------------|-------|---|-------|--|-------|----------------------|-------|----------------------|-------|----------|-------|---------|-------|----------------------|-------|----------------------|-------|----------------------|-------|----------------------|-------|----------------------|-------|----------------------|----|-----|
| 19:10 | autopchen  | <p>Auto-Precharge Enable: One bit is assigned to each control port. For each bit, the encodings are as follows:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0x0</td> <td>The controller requests an automatic precharge following a bus command completion (close the row automatically)</td> </tr> <tr> <td>0x1</td> <td>The controller attempts to keep a row open. All active ports with random dominated operations should set the autopchen bit to 1.</td> </tr> </tbody> </table> <p>The bits in this field correspond to the control ports as follows:</p> <table border="1"> <tbody> <tr> <td>Bit 9</td> <td>CPU write</td> </tr> <tr> <td>Bit 8</td> <td>L3 write</td> </tr> <tr> <td>Bit 7</td> <td>CPU read</td> </tr> <tr> <td>Bit 6</td> <td>L3 read</td> </tr> <tr> <td>Bit 5</td> <td>FPGA-to-SDRAM port 5</td> </tr> <tr> <td>Bit 4</td> <td>FPGA-to-SDRAM port 4</td> </tr> <tr> <td>Bit 3</td> <td>FPGA-to-SDRAM port 3</td> </tr> <tr> <td>Bit 2</td> <td>FPGA-to-SDRAM port 2</td> </tr> <tr> <td>Bit 1</td> <td>FPGA-to-SDRAM port 1</td> </tr> <tr> <td>Bit 0</td> <td>FPGA-to-SDRAM port 0</td> </tr> </tbody> </table> | Value  | Description          | 0x0   | The controller requests an automatic precharge following a bus command completion (close the row automatically) | 0x1   | The controller attempts to keep a row open. All active ports with random dominated operations should set the autopchen bit to 1. | Bit 9 | CPU write            | Bit 8 | L3 write             | Bit 7 | CPU read | Bit 6 | L3 read | Bit 5 | FPGA-to-SDRAM port 5 | Bit 4 | FPGA-to-SDRAM port 4 | Bit 3 | FPGA-to-SDRAM port 3 | Bit 2 | FPGA-to-SDRAM port 2 | Bit 1 | FPGA-to-SDRAM port 1 | Bit 0 | FPGA-to-SDRAM port 0 | RW | 0x0 |
| Value | Description  |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 0x0   | The controller requests an automatic precharge following a bus command completion (close the row automatically)                  |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 0x1   | The controller attempts to keep a row open. All active ports with random dominated operations should set the autopchen bit to 1. |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 9 | CPU write  |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 8 | L3 write   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 7 | CPU read   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 6 | L3 read  |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 5 | FPGA-to-SDRAM port 5   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 4 | FPGA-to-SDRAM port 4   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 3 | FPGA-to-SDRAM port 3   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 2 | FPGA-to-SDRAM port 2   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 1 | FPGA-to-SDRAM port 1   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 0 | FPGA-to-SDRAM port 0   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 5:0   | portprotocol   | <p>Port Protocol: You can read this field to determine the protocol configuration of each of the FPGA-to-SDRAM ports. The bits in this field correspond to the control ports as follows:</p> <table border="1"> <tbody> <tr> <td>Bit 5</td> <td>FPGA-to-SDRAM port 5</td> </tr> <tr> <td>Bit 4</td> <td>FPGA-to-SDRAM port 4</td> </tr> <tr> <td>Bit 3</td> <td>FPGA-to-SDRAM port 3</td> </tr> <tr> <td>Bit 2</td> <td>FPGA-to-SDRAM port 2</td> </tr> <tr> <td>Bit 1</td> <td>FPGA-to-SDRAM port 1</td> </tr> </tbody> </table>  | Bit 5  | FPGA-to-SDRAM port 5 | Bit 4 | FPGA-to-SDRAM port 4  | Bit 3 | FPGA-to-SDRAM port 3   | Bit 2 | FPGA-to-SDRAM port 2 | Bit 1 | FPGA-to-SDRAM port 1 | RO    | 0x0      |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 5 | FPGA-to-SDRAM port 5   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 4 | FPGA-to-SDRAM port 4   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 3 | FPGA-to-SDRAM port 3   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 2 | FPGA-to-SDRAM port 2   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 1 | FPGA-to-SDRAM port 1   |  |        |                      |       |   |       |  |       |                      |       |                      |       |          |       |         |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |

*continued...*



| Bit   | Name                            | Description  | Access | Reset                       |     |                                 |     |           |  |  |
|-------|---------------------------------|--|--------|-----------------------------|-----|---------------------------------|-----|-----------|--|--|
|       |                                 | <p>Bit 0   FPGA-to-SDRAM port 0</p> <p>When you read the corresponding port bit after the FPGA has been configured, it will have one of the following values:</p> <table border="1"><thead><tr><th>Value</th><th>Protocol Configuration Type</th></tr></thead><tbody><tr><td>0x1</td><td>AXI (reset value for all ports)</td></tr><tr><td>0x0</td><td>Avalon-MM</td></tr></tbody></table> <p><i>Note:</i> The value in this field is only valid after the fabric has been configured.</p> <p><i>Note:</i> The AXI protocol requires both a read and a write port. Therefore, you must ensure that AXI ports are allocated in the pairs (port 0, port 1), (port 2, port 3), and (port 4, port 5).</p> <p>Aside from the requirement noted above, AXI and Avalon-MM ports can be mixed freely.</p> | Value  | Protocol Configuration Type | 0x1 | AXI (reset value for all ports) | 0x0 | Avalon-MM |  |  |
| Value | Protocol Configuration Type     |  |        |                             |     |                                 |     |           |  |  |
| 0x1   | AXI (reset value for all ports) |  |        |                             |     |                                 |     |           |  |  |
| 0x0   | Avalon-MM                       |  |        |                             |     |                                 |     |           |  |  |

#### 7.15.1.1.22. fpgaportrst

This register implements functionality to allow the CPU to control when the MPFE will enable the ports to the FPGA fabric.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25080       |

Offset: 0x5080

Access: RW

*Important:* To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |                    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
|------------|--------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 31         | 30                 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| Reserved   |                    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 15         | 14                 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| Reserved   | portrstn<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |

#### fpgaportrst Fields

| Bit  | Name     | Description  | Access | Reset |
|------|----------|--|--------|-------|
| 13:0 | portrstn | This register should be written to with a 1 to enable the selected FPGA port to exit reset. Writing a bit to a zero will stretch the port reset until the register is written. Read data ports are connected to bits 3:0, with read data port 0 at bit 0 to read data port 3 at bit 3. Write data ports 0 to 3 are mapped to 4 to 7, with write data port 0 connected to bit 4 to write data port 3 at bit 7. Command ports are connected to | RW     | 0x0   |



| Bit | Name | Description  | Access | Reset |
|-----|------|--|--------|-------|
|     |      | bits 8 to 13, with command port 0 at bit 8 to command port 5 at bit 13. Expected usage would be to set all the bits at the same time but setting some bits to a zero and others to a one is supported. |        |       |

#### 7.15.1.1.23. protportdefault

This register controls the default protection assignment for a port. Ports which have explicit rules which define regions which are illegal to access should set the bits to pass by default. Ports which have explicit rules which define legal areas should set the bit to force all transactions to fail. Leaving this register to all zeros should be used for systems which do not desire any protection from the memory controller.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2508C       |

Offset: 0x508C

Access: RW

*Important:* To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |                       |    |    |    |    |    |    |    |    |  |
|------------|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|----|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24                    | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| Reserved   |    |    |    |    |    |    |                       |    |    |    |    |    |    |    |    |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8                     | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| Reserved   |    |    |    |    |    |    | portdefault<br>RW 0x0 |    |    |    |    |    |    |    |    |  |

#### protportdefault Fields

| Bit   | Name                           | Description  | Access | Reset     |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
|-------|--------------------------------|--|--------|-----------|-------|----------|-------|----------|-------|---------|-------|--------------------------------|-------|--------------------------------|-------|--------------------------------|-------|--------------------------------|-------|--------------------------------|----|-----|
| 9:0   | portdefault                    | <p>Determines the default action for specified transactions. When a bit is zero, the specified access is allowed by default. When a bit is one, the specified access is denied by default.</p> <table border="1"> <tr> <td>Bit 9</td> <td>CPU write</td> </tr> <tr> <td>Bit 8</td> <td>L3 write</td> </tr> <tr> <td>Bit 7</td> <td>CPU read</td> </tr> <tr> <td>Bit 6</td> <td>L3 read</td> </tr> <tr> <td>Bit 5</td> <td>Access to FPGA-to-SDRAM port 5</td> </tr> <tr> <td>Bit 4</td> <td>Access to FPGA-to-SDRAM port 4</td> </tr> <tr> <td>Bit 3</td> <td>Access to FPGA-to-SDRAM port 3</td> </tr> <tr> <td>Bit 2</td> <td>Access to FPGA-to-SDRAM port 2</td> </tr> <tr> <td>Bit 1</td> <td>Access to FPGA-to-SDRAM port 1</td> </tr> </table> | Bit 9  | CPU write | Bit 8 | L3 write | Bit 7 | CPU read | Bit 6 | L3 read | Bit 5 | Access to FPGA-to-SDRAM port 5 | Bit 4 | Access to FPGA-to-SDRAM port 4 | Bit 3 | Access to FPGA-to-SDRAM port 3 | Bit 2 | Access to FPGA-to-SDRAM port 2 | Bit 1 | Access to FPGA-to-SDRAM port 1 | RW | 0x0 |
| Bit 9 | CPU write                      |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 8 | L3 write                       |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 7 | CPU read                       |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 6 | L3 read                        |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 5 | Access to FPGA-to-SDRAM port 5 |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 4 | Access to FPGA-to-SDRAM port 4 |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 3 | Access to FPGA-to-SDRAM port 3 |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 2 | Access to FPGA-to-SDRAM port 2 |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |
| Bit 1 | Access to FPGA-to-SDRAM port 1 |  |        |           |       |          |       |          |       |         |       |                                |       |                                |       |                                |       |                                |       |                                |    |     |



| Bit | Name | Description |                                | Access | Reset |
|-----|------|-------------|--------------------------------|--------|-------|
|     |      | Bit 0       | Access to FPGA-to-SDRAM port 0 |        |       |

#### 7.15.1.1.24. protruleaddr

This register is used to control the memory protection for port 0 transactions. Address ranges can either be used to allow access to memory regions or disallow access to memory regions. If TrustZone is being used, access can be enabled for protected transactions or disabled for unprotected transactions. The default state of this register is to allow all access. Address values used for protection are only physical addresses.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25090       |

Offset: 0x5090

Access: RW

*Important:* To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields         |    |    |    |    |    |    |    |                   |    |                    |    |    |    |    |    |  |  |
|--------------------|----|----|----|----|----|----|----|-------------------|----|--------------------|----|----|----|----|----|--|--|
| 31                 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23                | 22 | 21                 | 20 | 19 | 18 | 17 | 16 |  |  |
| Reserved           |    |    |    |    |    |    |    |                   |    | highaddr<br>RW 0x0 |    |    |    |    |    |  |  |
| 15                 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7                 | 6  | 5                  | 4  | 3  | 2  | 1  | 0  |  |  |
| highaddr<br>RW 0x0 |    |    |    |    |    |    |    | lowaddr<br>RW 0x0 |    |                    |    |    |    |    |    |  |  |

#### protruleaddr Fields

| Bit   | Name     | Description  | Access | Reset |
|-------|----------|--|--------|-------|
| 23:12 | highaddr | Upper 12 bits of the address for a check. Address is compared to be greater than or equal to the address of a transaction. Note that since AXI transactions cannot cross a 4K byte boundary, the transaction start and transaction end address must also fall within the same 1MByte block pointed to by this address pointer. | RW     | 0x0   |
| 11:0  | lowaddr  | Lower 12 bits of the address for a check. Address is compared to be less than or equal to the address of a transaction. Note that since AXI transactions cannot cross a 4K byte boundary, the transaction start and transaction end address must also fall within the same 1MByte block pointed to by this address pointer.    | RW     | 0x0   |

#### 7.15.1.1.25. protruleid

This register configures the AxID for a given protection rule.



| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25094       |

Offset: 0x5094

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields       |    |    |    |                 |    |    |    |                  |    |    |    |    |    |    |    |  |
|------------------|----|----|----|-----------------|----|----|----|------------------|----|----|----|----|----|----|----|--|
| 31               | 30 | 29 | 28 | 27              | 26 | 25 | 24 | 23               | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| Reserved         |    |    |    |                 |    |    |    | highid<br>RW 0x0 |    |    |    |    |    |    |    |  |
| 15               | 14 | 13 | 12 | 11              | 10 | 9  | 8  | 7                | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| highid<br>RW 0x0 |    |    |    | lowid<br>RW 0x0 |    |    |    |                  |    |    |    |    |    |    |    |  |

### protruleid Fields

| Bit       | Name   | Description   | Access | Reset |
|-----------|--------|---|--------|-------|
| 23:1<br>2 | highid | AxID for the protection rule. Incoming AxID needs to be less than or equal to this value. For all AxIDs from a port, AxID high should be programmed to all ones.    | RW     | 0x0   |
| 11:0      | lowid  | AxID for the protection rule. Incoming AxID needs to be greater than or equal to this value. For all AxIDs from a port, AxID high should be programmed to all ones. | RW     | 0x0   |

### 7.15.1.1.26. protruledata

This register configures the protection memory characteristics of each protection rule.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC25098       |

Offset: 0x5098

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.



| Bit Fields |                              |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                    |                            |                    |
|------------|------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------------|----------------------------|--------------------|
| 31         | 30                           | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |                    |                            |                    |
| Reserved   |                              |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                    |                            |                    |
| 15         | 14                           | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |                    |                            |                    |
| Reserved   | ruler<br>result<br>RW<br>0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    | portmask<br>RW 0x0 | valid<br>rule<br>RW<br>0x0 | security<br>RW 0x0 |

### protruledata Fields

| Bit        | Name   | Description  | Access | Reset       |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
|------------|--|--|--------|-------------|-------|-------------------------------------|-------|---|------------|--|-------|----------------------|-------|----------------------|-------|----------------------|-------|----------------------|-------|----------------------|-------|----------------------|----|-----|
| 13         | rulerresult  | Set this bit to a one to force a protection failure, zero to allow the access to succeed   | RW     | 0x0         |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 12:3       | portmask   | The bits in this field determine which ports the rule applies to. If a port's bit is set, the rule applies to that port; if the bit is clear, the rule does not apply. The bits in this field correspond to the control ports as follows:<br><br><table border="1"><tr><td>Bit 9</td><td>CPU write</td></tr><tr><td>Bit 8</td><td>L3 write</td></tr><tr><td>Bit 7</td><td>CPU read</td></tr><tr><td>Bit 6</td><td>L3 read</td></tr><tr><td>Bit 5</td><td>FPGA-to-SDRAM port 5</td></tr><tr><td>Bit 4</td><td>FPGA-to-SDRAM port 4</td></tr><tr><td>Bit 3</td><td>FPGA-to-SDRAM port 3</td></tr><tr><td>Bit 2</td><td>FPGA-to-SDRAM port 2</td></tr><tr><td>Bit 1</td><td>FPGA-to-SDRAM port 1</td></tr><tr><td>Bit 0</td><td>FPGA-to-SDRAM port 0</td></tr></table><br>& | Bit 9  | CPU write   | Bit 8 | L3 write                            | Bit 7 | CPU read                                | Bit 6      | L3 read  | Bit 5 | FPGA-to-SDRAM port 5 | Bit 4 | FPGA-to-SDRAM port 4 | Bit 3 | FPGA-to-SDRAM port 3 | Bit 2 | FPGA-to-SDRAM port 2 | Bit 1 | FPGA-to-SDRAM port 1 | Bit 0 | FPGA-to-SDRAM port 0 | RW | 0x0 |
| Bit 9      | CPU write  |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 8      | L3 write   |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 7      | CPU read   |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 6      | L3 read  |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 5      | FPGA-to-SDRAM port 5                               |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 4      | FPGA-to-SDRAM port 4                               |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 3      | FPGA-to-SDRAM port 3                               |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 2      | FPGA-to-SDRAM port 2                               |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 1      | FPGA-to-SDRAM port 1                               |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Bit 0      | FPGA-to-SDRAM port 0                               |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 2          | validrule  | Set to bit to a one to make a rule valid, set to a zero to invalidate a rule.  | RW     | 0x0         |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 1:0        | security   | Valid security field encodings are:<br><br><table border="1"><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>0x0</td><td>Rule applies to secure transactions</td></tr><tr><td>0x1</td><td>Rule applies to non-secure transactions</td></tr><tr><td>0x2 or 0x3</td><td>Rule applies to secure and non-secure transactions</td></tr></tbody></table>  | Value  | Description | 0x0   | Rule applies to secure transactions | 0x1   | Rule applies to non-secure transactions | 0x2 or 0x3 | Rule applies to secure and non-secure transactions | RW    | 0x0                  |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| Value      | Description  |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 0x0        | Rule applies to secure transactions                |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 0x1        | Rule applies to non-secure transactions            |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |
| 0x2 or 0x3 | Rule applies to secure and non-secure transactions |  |        |             |       |                                     |       |   |            |  |       |                      |       |                      |       |                      |       |                      |       |                      |       |                      |    |     |

### 7.15.1.1.27. protrulerdwr

This register is used to perform read and write operations to the internal protection table.



| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC2509C       |

Offset: 0x509C

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |    |    |          |           |                      |    |    |    |          |
|------------|----|----|----|----|----|----|----|----|----|----------|-----------|----------------------|----|----|----|----------|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21       | 20        | 19                   | 18 | 17 | 16 | Reserved |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5        | 4         | 3                    | 2  | 1  | 0  | Reserved |
|            |    |    |    |    |    |    |    |    |    | readrule | writerule | ruleoffset<br>RW 0x0 |    |    |    |          |

### protrulerdwr Fields

| Bit | Name       | Description   | Access | Reset |
|-----|------------|---|--------|-------|
| 6   | readrule   | Write to this bit to have the memory_prot_data register loaded with the value from the internal protection table at offset. Table value will be loaded before a rd is returned so read data from the register will be correct for any follow-on reads to the memory_prot_data register. | RW     | 0x0   |
| 5   | writerule  | Write to this bit to have the memory_prot_data register to the table at the offset specified by port_offset. Bit automatically clears after a single cycle and the write operation is complete.   | RW     | 0x0   |
| 4:0 | ruleoffset | This field defines which of the 20 rules in the protection table you want to read or write.   | RW     | 0x0   |

#### 7.15.1.1.28. mppriority

This register is used to configure the DRAM burst operation scheduling.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC250AC       |

Offset: 0x50AC

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.



| Bit Fields             |    |                        |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
|------------------------|----|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 31                     | 30 | 29                     | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| Reserved               |    | userpriority<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 15                     | 14 | 13                     | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| userpriority<br>RW 0x0 |    |                        |    |    |    |    |    |    |    |    |    |    |    |    |    |  |

### mppriority Fields

| Bit  | Name         | Description  | Access | Reset |
|------|--------------|--|--------|-------|
| 29:0 | userpriority | User Priority: This field sets the absolute user priority of each port, which is represented as a 3-bit value. 0x0 is the lowest priority and 0x7 is the highest priority. Port 0 is configured by programming userpriority[2:0], port 1 is configured by programming userpriority[5:3], port 2 is configured by programming userpriority[8:6], and so on. | RW     | 0x0   |

#### 7.15.1.1.29. remappriority

This register applies another level of port priority after a transaction is placed in the single port queue.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC250E0       |

Offset: 0x50E0

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields |    |    |    |    |    |    |    |                         |    |    |    |    |    |    |    |  |
|------------|----|----|----|----|----|----|----|-------------------------|----|----|----|----|----|----|----|--|
| 31         | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23                      | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| Reserved   |    |    |    |    |    |    |    |                         |    |    |    |    |    |    |    |  |
| 15         | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7                       | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| Reserved   |    |    |    |    |    |    |    | priorityremap<br>RW 0x0 |    |    |    |    |    |    |    |  |

### remappriority Fields

| Bit | Name          | Description  | Access | Reset |
|-----|---------------|--|--------|-------|
| 7:0 | priorityremap | Each bit of this field represents a priority level. If bit N in the priorityremap field is set, then any port transaction with absolute user priority of N jumps to the front of the single port queue and is serviced ahead of any transactions in the queue. For example, if bit 5 is set in the priorityremap field of the remappriority register, then any port transaction with a userpriority value of 0x5 in the mppriority register is serviced ahead of any other transaction already in the single port queue. | RW     | 0x0   |



### 7.15.1.1.30. Port Sum of Weight Register Register Descriptions

This register is used to configure the DRAM burst operation scheduling.

Offset: 0xb0

- [mpweight\\_0\\_4](#) on page 333
- [mpweight\\_1\\_4](#) on page 333
- [mpweight\\_2\\_4](#) on page 334
- [mpweight\\_3\\_4](#) on page 334

#### [mpweight\\_0\\_4](#)

This register is used to configure the DRAM burst operation scheduling.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC250B0       |

Offset: 0x50B0

Access: RW

| Bit Fields        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |        |
|-------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|
| 31                | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |        |
| staticweight_31_0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |        |
|                   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RW 0x0 |
| staticweight_31_0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |        |
|                   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RW 0x0 |

#### [mpweight\\_0\\_4](#) Fields

| Bit  | Name              | Description  | Access | Reset |
|------|-------------------|--|--------|-------|
| 31:0 | staticweight_31_0 | Set static weight of the port. Each port is programmed with a 5 bit value. Port 0 is bits 4:0, port 1 is bits 9:5, up to port 9 being bits 49:45 | RW     | 0x0   |

#### [mpweight\\_1\\_4](#)

This register is used to configure the DRAM burst operation scheduling.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC250B4       |

Offset: 0x50B4

Access: RW



| Bit Fields                   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                              |  |
|------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------------|--|
| 31                           | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16                           |  |
| sumofweights_13_0<br>RW 0x0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    | staticweight_49_32<br>RW 0x0 |  |
| 15                           | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0                            |  |
| staticweight_49_32<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                              |  |

### mpweight\_1\_4 Fields

| Bit       | Name               | Description   | Access | Reset |
|-----------|--------------------|---|--------|-------|
| 31:1<br>8 | sumofweights_13_0  | Set the sum of static weights for particular user priority. This register is used as part of the deficit round robin implementation. It should be set to the sum of the weights for the ports | RW     | 0x0   |
| 17:0      | staticweight_49_32 | Set static weight of the port. Each port is programmed with a 5 bit value. Port 0 is bits 4:0, port 1 is bits 9:5, up to port 9 being bits 49:45  | RW     | 0x0   |

### mpweight\_2\_4

This register is used to configure the DRAM burst operation scheduling.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC250B8       |

Offset: 0x50B8

Access: RW

| Bit Fields                   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
|------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 31                           | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |  |
| sumofweights_45_14<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |
| 15                           | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |  |
| sumofweights_45_14<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |

### mpweight\_2\_4 Fields

| Bit  | Name               | Description   | Access | Reset |
|------|--------------------|---|--------|-------|
| 31:0 | sumofweights_45_14 | Set the sum of static weights for particular user priority. This register is used as part of the deficit round robin implementation. It should be set to the sum of the weights for the ports | RW     | 0x0   |

### mpweight\_3\_4

This register is used to configure the DRAM burst operation scheduling.

| Module Instance | Base Address | Register Address |
|-----------------|--------------|------------------|
| sdr             | 0xFFC20000   | 0xFFC250BC       |



Offset: 0x50BC

Access: RW

**Important:** To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

| Bit Fields                   |    |    |    |    |    |    |    |    |    |    |    |    |    |                              |    |  |
|------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------------|----|--|
| 31                           | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17                           | 16 |  |
| Reserved                     |    |    |    |    |    |    |    |    |    |    |    |    |    | sumofweights_63_46<br>RW 0x0 |    |  |
| 15                           | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1                            | 0  |  |
| sumofweights_63_46<br>RW 0x0 |    |    |    |    |    |    |    |    |    |    |    |    |    |                              |    |  |

#### mpweight\_3\_4 Fields

| Bit  | Name               | Description   | Access | Reset |
|------|--------------------|---|--------|-------|
| 17:0 | sumofweights_63_46 | Set the sum of static weights for particular user priority. This register is used as part of the deficit round robin implementation. It should be set to the sum of the weights for the ports | RW     | 0x0   |

## 7.16. Document Revision History

| Date          | Version    | Changes   |
|---------------|------------|---|
| May 2017      | 2017.05.08 | <ul style="list-style-type: none"> <li>Added note to first topic.</li> <li>Rebranded as Intel.</li> </ul>   |
| October 2016  | 2016.10.28 | Maintenance release   |
| May 2016      | 2016.05.03 | Maintenance release   |
| November 2015 | 2015.11.02 | <ul style="list-style-type: none"> <li>Added information regarding calculation of ECC error byte address location from erraddr register in "User Notification of ECC Errors" sectoin</li> <li>Added information regarding bus response to memory protection transaction failure in "Memory Protection" section</li> <li>Clarified "Protection" row in "Fields for Rules in Memory Protection" table in the "Memory Protection" section</li> <li>Clarified protruledata.security column in "Rules in Memory Protection Table for Example Configuration" table in the "Example of Configuration for TrustZone" section</li> <li>Added note about double-bit error functionality in "ECC Write Backs" subsection of "ECC" section</li> <li>Added the "DDR Calibration" subsection under "DDR PHY" section</li> </ul> |
| May 2015      | 2015.05.04 | <ul style="list-style-type: none"> <li>Added the recommended sequence for writing or reading a rule in the "Memory Protection" section.</li> </ul>  |

*continued...*



| Date          | Version    | Changes  |
|---------------|------------|--|
| December 2014 | 2014.12.15 | <ul style="list-style-type: none"><li>Added SDRAM Protection Access Flow Diagram to "Memory Protection" subsection in the "Single-Port Controller Operation" section.</li><li>Changed the "SDRAM Multi-Port Scheduling" section to "SDRAM Multi-Port Arbitration" and added detailed information on how to use and program the priority and weighted arbitration scheme.</li></ul> |
| June 2014     | 2014.6.30  | <ul style="list-style-type: none"><li>Added <i>Port Mappings</i> section.</li><li>Added <i>SDRAM Controller Memory Options</i> section.</li><li>Enhanced <i>Example of Configuration for TrustZone</i> section.</li><li>Added SDRAM Controller address map and registers.</li></ul>  |
| December 2013 | 2013.12.30 | <ul style="list-style-type: none"><li>Added <i>Generating a Preloader Image for HPS with EMIF</i> section.</li><li>Added <i>Debugging HPS SDRAM in the Preloader</i> section.</li><li>Enhanced <i>Simulation</i> section.</li></ul>  |
| November 2012 | 1.1        | Added address map and register definitions section.  |
| January 2012  | 1.0        | Initial release.   |

## 8. Functional Description—HPC II Controller

The High Performance Controller II works with the UniPHY-based DDR2, DDR3, and LPDDR2 interfaces. The controller provides high memory bandwidth, high clock rate performance, and run-time programmability. The controller can reorder data to reduce row conflicts and bus turn-around time by grouping reads and writes together, allowing for efficient traffic patterns and reduced latency.

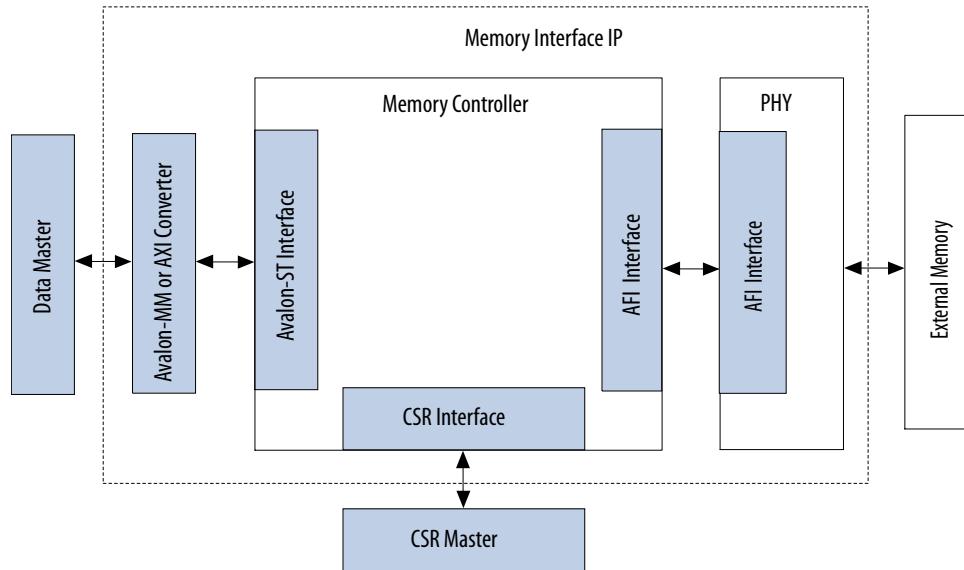
**Note:** The controller described here is the High Performance Controller II (HPC II) with advanced features for designs generated in the Quartus II software version 11.0 and later, and the Quartus Prime software. Designs created in earlier versions and regenerated in version 11.0 and later do not inherit the new advanced features; for information on HPC II without the version 11.0 and later advanced features, refer to the External Memory Interface Handbook for Quartus II version 10.1, available in the External Memory Interfaces literature section on [www.altera.com](http://www.altera.com).

### Related Information

[External Memory Interface Handbook, v10.1](#)

### 8.1. HPC II Memory Interface Architecture

The memory interface consists of the memory controller logic block, the physical logic layer (PHY), and their associated interfaces. The following figure shows a high-level block diagram of the overall external memory interface architecture.

**Figure 156. High-Level Diagram of Memory Interface Architecture**

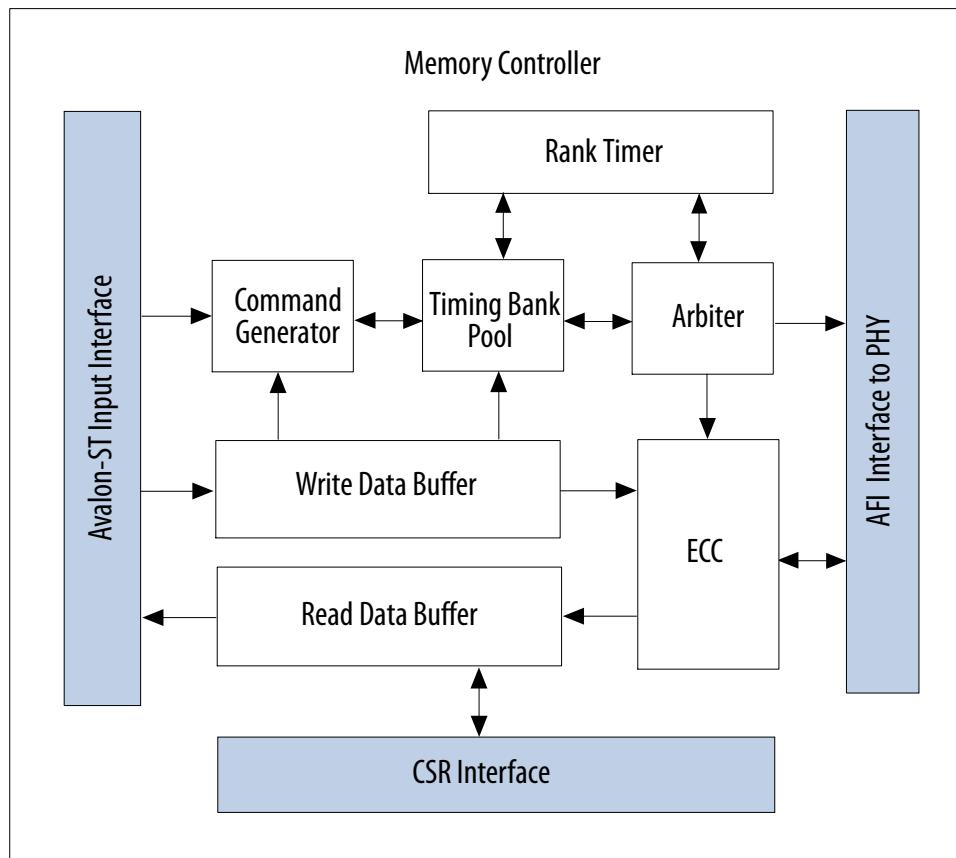
## 8.2. HPC II Memory Controller Architecture

The memory controller logic block uses an Avalon Streaming (Avalon-ST) interface as its native interface, and communicates with the PHY layer by the Altera PHY Interface (AFI).

The following figure shows a block diagram of the memory controller architecture.



Figure 157. Memory Controller Architecture Block Diagram



#### Avalon-ST Input Interface

The Avalon-ST interface serves as the entry point to the memory controller, and provides communication with the requesting data masters.

For information about the Avalon interface, refer to *Avalon Interface Specifications*.

#### AXI to Avalon-ST Converter

The HPC II memory controller includes an AXI to Avalon-ST converter for communication with the AXI protocol. The AXI to Avalon-ST converter provides write address, write data, write response, read address, and read data channels on the AXI interface side, and command, write data, and read data channels on the Avalon-ST interface side.

#### Handshaking

The AXI protocol employs a handshaking process similar to the Avalon-ST protocol, based on ready and valid signals.

#### Command Channel Implementation

The AXI interface includes separate read and write channels, while the Avalon-ST interface has only one command channel. Arbitration of the read and write channels is based on these policies:

- Round robin
- Write priority—write channel has priority over read channel
- Read priority—read channel has priority over write channel

You can choose an arbitration policy by setting the `COMMAND_ARB_TYPE` parameter to one of `ROUND_ROBIN`, `WRITE_PRIORITY`, or `READ_PRIORITY` in the `alt_mem_ddrx_axi_st_converter.v` file.

### Data Ordering

The AXI specification requires that write data IDs must arrive in the same order as write address IDs are received. Similarly, read data must be returned in the same order as its associated read address is received.

Consequently, the AXI to Avalon-ST converter does not support interleaving of write data; all data must arrive in the same order as its associated write address IDs. On the read side, the controller returns read data based on the read addresses received.

### Burst Types

The AXI to Avalon-ST converter supports the following burst types:

- Incrementing burst—the address for each transfer is an increment of the previous transfer address; the increment value depends on the size of the transfer.
- Wrapping burst—similar to the incrementing burst, but wraps to the lower address when the burst boundary is reached. The starting address must be aligned to the size of the transfer. Burst length must be 2, 4, 8, or 16. The burst wrap boundary = burst size \* burst length.

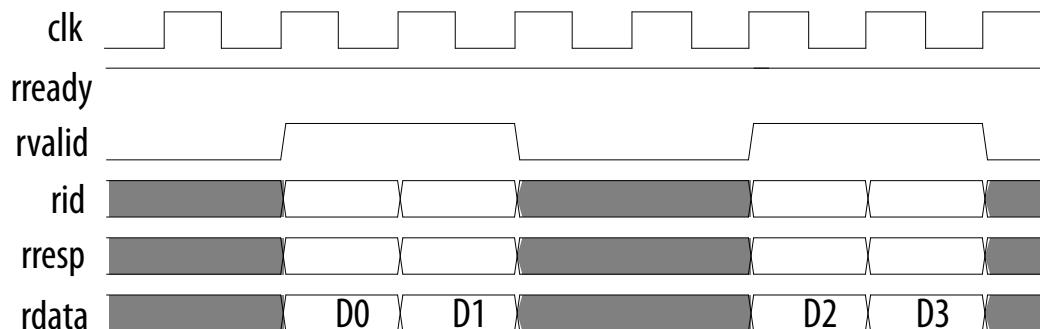
### Related Information

[Avalon Interface Specifications](#)

## 8.2.1. Backpressure Support

The write response and read data channels do not support data transfer with backpressure; consequently, you must assert the ready signal for the write response and read data channels to 1 to ensure acceptance of data at any time.

**Figure 158. Data Transfer Without Backpressure**





For information about data transfer with and without backpressure, refer to the *Avalon Interface Specifications*.

#### Related Information

[Avalon Interface Specifications](#)

### 8.2.2. Command Generator

The command generator accepts commands from the front-end Avalon-ST interface and from local ECC internal logic, and provides those commands to the timing bank pool.

### 8.2.3. Timing Bank Pool

The timing bank pool is a parallel queue that works with the arbiter to enable data reordering. The timing bank pool tracks incoming requests, ensures that all timing requirements are met and, upon receiving write-data-ready notification from the write data buffer, passes the requests to the arbiter in an ordered and efficient manner.

### 8.2.4. Arbiter

The arbiter determines the order in which requests are passed to the memory device. When the arbiter receives a single request, that request is passed immediately; however, when multiple requests are received, the arbiter uses arbitration rules to determine the order in which to pass requests to the memory device.

#### Arbitration Rules

The arbiter uses the following arbitration rules:

- If only one master is issuing a request, grant that request immediately.
- If there are outstanding requests from two or more masters, the arbiter applies the following tests, in order:
  - Is there a read request? If so, the arbiter grants the read request ahead of any write requests.
  - If neither of the above conditions apply, the arbiter grants the oldest request first.

### 8.2.5. Rank Timer

The rank timer maintains rank-specific timing information, and performs the following functions:

- Ensures that only four activates occur within a specified timing window.
- Manages the read-to-write and write-to-read bus turnaround time.
- Manages the time-to-activate delay between different banks.

### 8.2.6. Read Data Buffer and Write Data Buffer

The read data buffer receives data from the PHY and passes that data through the input interface to the master. The write data buffer receives write data from the input interface and passes that data to the PHY, upon approval of the write request.



### 8.2.7. ECC Block

The error-correcting code (ECC) block comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC block can remedy errors resulting from noise or other impairments during data transmission.

### 8.2.8. AFI and CSR Interfaces

The AFI interface provides communication between the controller and the physical layer logic (PHY). The CSR interface provides communication with your system's internal control status registers.

For more information about AFI signals, refer to *AFI 4.0 Specification* in the *Functional Description - UniPHY chapter*.

*Note:* Unaligned reads and writes on the AFI interface are not supported.

#### Related Information

- [Avalon Interface Specifications](#)
- [Functional Description—UniPHY](#) on page 13

## 8.3. HPC II Controller Features

The HPC II memory controller offers a variety of features.

### 8.3.1. Data Reordering

The controller implements data reordering to maximize efficiency for read and write commands. The controller can reorder read and write commands as necessary to mitigate bus turn-around time and reduce conflict between rows.

Inter-bank data reordering reorders commands going to different bank addresses. Commands going to the same bank address are not reordered. This reordering method implements simple hazard detection on the bank address level.

The controller implements logic to limit the length of time that a command can go unserved. This logic is known as starvation control. In starvation control, a counter is incremented for every command served. You can set a starvation limit, to ensure that a waiting command is served immediately, when the starvation counter reaches the specified limit.

### 8.3.2. Pre-emptive Bank Management

Data reordering allows the controller to issue bank-management commands pre-emptively, based on the patterns of incoming commands. The desired page in memory can be already open when a command reaches the AFI interface.



### 8.3.3. Quasi-1T and Quasi-2T

One controller clock cycle equals two memory clock cycles in a half-rate interface, and to four memory clock cycles in a quarter-rate interface. To fully utilize the command bandwidth, the controller can operate in Quasi-1T half-rate and Quasi-2T quarter-rate modes.

In Quasi-1T and Quasi-2T modes, the controller issues two commands on every controller clock cycle. The controller is constrained to issue a row command on the first clock phase and a column command on the second clock phase, or vice versa. Row commands include activate and precharge commands; column commands include read and write commands.

The controller operates in Quasi-1T in half-rate mode, and in Quasi-2T in quarter-rate mode; this operation is transparent and has no user settings.

### 8.3.4. User Autoprecharge Commands

The autoprecharge read and autoprecharge write commands allow you to indicate to the memory device that this read or write command is the last access to the currently open row. The memory device automatically closes or autoprecharges the page it is currently accessing so that the next access to the same bank is quicker.

This command is useful for applications that require fast random accesses.

Since the HPC II controller can reorder transactions for best efficiency, when you assert the `local_autopch_req` signal, the controller evaluates the current command and buffered commands to determine the best autoprecharge operation.

### 8.3.5. Address and Command Decoding Logic

When the main state machine issues a command to the memory, it asserts a set of internal signals. The address and command decoding logic turns these signals into AFI-specific commands and address.

The following signals are generated:

- Clock enable and reset signals: `afi_cke`, `afi_rst_n`
- Command and address signals: `afi_cs_n`, `afi_ba`, `afi_addr`, `afi_ras_n`, `afi_cas_n`, `afi_we_n`

### 8.3.6. Low-Power Logic

There are two types of low-power logic: the user-controlled self-refresh logic and automatic power-down with programmable time-out logic.

#### User-Controlled Self-Refresh

When you assert the `local_self_rfsh_req` signal, the controller completes any currently executing reads and writes, and then interrupts the command queue and immediately places the memory into self-refresh mode. When the controller places the memory into self-refresh mode, it responds by asserting an acknowledge signal, `local_self_rfsh_ack`. You can leave the memory in self-refresh mode for as long as you choose.



To bring the memory out of self-refresh mode, you must deassert the request signal, and the controller responds by deasserting the acknowledge signal when the memory is no longer in self-refresh mode.

- Note:* If a user-controlled refresh request and a system-generated refresh request occur at the same time, the user-controlled refresh takes priority; the system-generated refresh is processed only after the user-controlled refresh request is completed.

#### Automatic Power-Down with Programmable Time-Out

The controller automatically places the memory in power-down mode to save power if the requested number of idle controller clock cycles is observed in the controller. The **Auto Power Down Cycles** parameter on the **Controller Settings** tab allows you to specify a range between 1 to 65,535 idle controller clock cycles. The counter for the programmable time-out starts when there are no user read or write requests in the command queue. Once the controller places the memory in power-down mode, it responds by asserting the acknowledge signal, `local_power_down_ack`.

### 8.3.7. ODT Generation Logic

The on-die termination (ODT) generation logic generates the necessary ODT signals for the controller, based on the scheme that Intel recommends.

#### DDR2 SDRAM

- Note:* There is no ODT for reads.

**Table 82. ODT—DDR2 SDRAM Single Slot Single Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt [0] |

- Note:* There is no ODT for reads.

**Table 83. ODT—DDR2 SDRAM Single Slot Dual Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt [0] |
| mem_cs[1] | mem_odt [1] |

**Table 84. ODT—DDR2 SDRAM Dual Slot Single Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt [1] |
| mem_cs[1] | mem_odt [0] |



**Table 85. ODT—DDR2 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt[2]  |
| mem_cs[1] | mem_odt[3]  |
| mem_cs[2] | mem_odt[0]  |
| mem_cs[3] | mem_odt[1]  |

#### DDR3 SDRAM

*Note:* There is no ODT for reads.

**Table 86. ODT—DDR3 SDRAM Single Slot Single Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt[0]  |

*Note:* There is no ODT for reads.

**Table 87. ODT—DDR3 SDRAM Single Slot Dual Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt[0]  |
| mem_cs[1] | mem_odt[1]  |

**Table 88. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled               |
|-----------|---------------------------|
| mem_cs[0] | mem_odt[0] and mem_odt[1] |
| mem_cs[1] | mem_odt[0] and mem_odt[1] |

**Table 89. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Read)**

| Read On   | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt[1]  |
| mem_cs[1] | mem_odt[0]  |

**Table 90. ODT—DDR3 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

| Write On  | ODT Enabled               |
|-----------|---------------------------|
| mem_cs[0] | mem_odt[0] and mem_odt[2] |

*continued...*



| Write On  | ODT Enabled                |
|-----------|----------------------------|
| mem_cs[1] | mem_odt [1]and mem_odt [3] |
| mem_cs[2] | mem_odt [0]and mem_odt [2] |
| mem_cs[3] | mem_odt [1]and mem_odt [3] |

**Table 91. ODT—DDR3 SDRAM Dual Slot Dual Rank Per DIMM (Read)**

| Read On   | ODT Enabled |
|-----------|-------------|
| mem_cs[0] | mem_odt [2] |
| mem_cs[1] | mem_odt [3] |
| mem_cs[2] | mem_odt [0] |
| mem_cs[3] | mem_odt [1] |

### 8.3.8. Burst Merging

The burst merging feature improves controller efficiency by merging two burst chop commands of sequential addresses into one burst length command. Burst merging is opportunistic and happens when the controller receives commands faster than it can process (for example, Avalon commands of multiple burst length) or when the controller temporarily stops processing commands due to Refresh.

The burst merging feature is turned off by default when you generate a controller. If your traffic exercises patterns that you can merge, you should turn on burst merging, as follows:

1. In a text editor, open the `<variation_name>.c0.v` top-level file for your design.
2. Search for the `ENABLE_BURST_MERGE` parameter in the `.v` file.
3. Change the `ENABLE_BURST_MERGE` value from 0 to 1.

### 8.3.9. ECC

The ECC logic comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC logic is available in multiples of 16, 24, 40, and 72 bits.

**Note:** For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, ECC logic is limited to widths of 24 and 40.

- The ECC logic has the following features:
- Has Hamming code ECC logic that encodes every 64, 32, 16, or 8 bits of data into 72, 40, 24, or 16 bits of codeword.
- Has a latency increase of one clock for both writes and reads.
- For a 128-bit interface, ECC is generated as one 64-bit data path with 8-bits of ECC path, plus a second 64-bit data path with 8-bits of ECC path.
- Detects and corrects all single-bit errors.



- Detects all double-bit errors.
- Counts the number of single-bit and double-bit errors.
- Accepts partial writes, which trigger a read-modify-write cycle, for memory devices with DM pins.
- Can inject single-bit and double-bit errors to trigger ECC correction for testing and debugging purposes.
- Generates an interrupt signal when an error occurs.

**Note:** When using ECC, you must initialize your entire memory content to zero before beginning to write to the memory. If you do not initialize the content to zero, and if you read from uninitialized memory locations without having first written to them, you will see junk data which will trigger an ECC interrupt.

When a single-bit or double-bit error occurs, the ECC logic triggers the `ecc_interrupt` signal to inform you that an ECC error has occurred. When a single-bit error occurs, the ECC logic reads the error address, and writes back the corrected data. When a double-bit error occurs, the ECC logic does not do any error correction but it asserts the `avl_rdata_error` signal to indicate that the data is incorrect. The `avl_rdata_error` signal follows the same timing as the `avl_rdata_valid` signal.

Enabling autocorrection allows the ECC logic to delay all controller pending activities until the correction completes. You can disable autocorrection and schedule the correction manually when the controller is idle to ensure better system efficiency. To manually correct ECC errors, follow these steps:

1. When an interrupt occurs, read out the `SBE_ERROR` register. When a single-bit error occurs, the `SBE_ERROR` register is equal to one.
2. Read out the `ERR_ADDR` register.
3. Correct the single-bit error by issuing a dummy write to the memory address stored in the `ERR_ADDR` register. A dummy write is a write request with the `local_be` signal zero, that triggers a partial write which is effectively a read-modify-write event. The partial write corrects the data at that address and writes it back.

### 8.3.9.1. Partial Writes

The ECC logic supports partial writes.

Along with the address, data, and burst signals, the Avalon-MM interface also supports a signal vector, `local_be`, that is responsible for byte-enable. Every bit of this signal vector represents a byte on the data-bus. Thus, a logic low on any of these bits instructs the controller not to write to that particular byte, resulting in a partial write. The ECC code is calculated on all bytes of the data-bus. If any bytes are changed, the IP core must recalculate the ECC code and write the new code back to the memory.

For partial writes, the ECC logic performs the following steps:

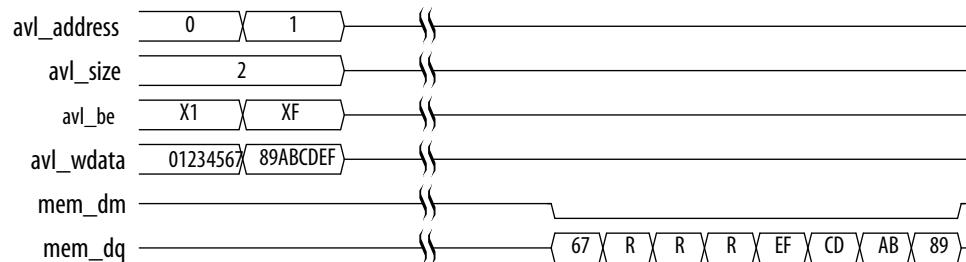
1. The ECC logic sends a read command to the partial write address.
2. Upon receiving a return data from the memory for the particular address, the ECC logic decodes the data, checks for errors, and then merges the corrected or correct dataword with the incoming information.
3. The ECC logic issues a write to write back the updated data and the new ECC code.

The following corner cases can occur:

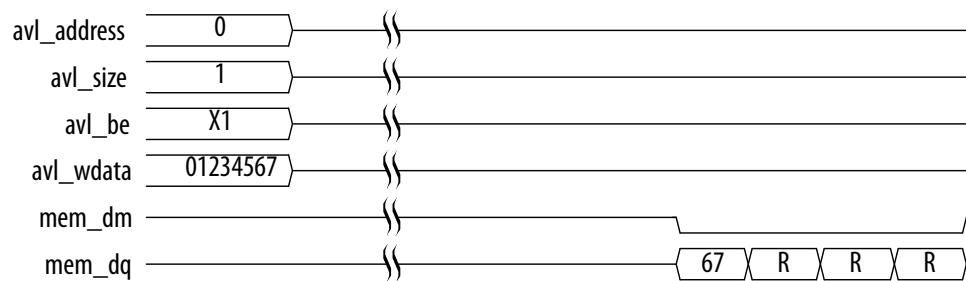
- A single-bit error during the read phase of the read-modify-write process. In this case, the IP core corrects the single-bit error first, increments the single-bit error counter and then performs a partial write to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the IP core increments the double-bit error counter and issues an interrupt. The IP core writes a new write word to the location of the error. The ECC status register keeps track of the error information.

The following figures show partial write operations for the controller, for full and half rate configurations, respectively.

**Figure 159. Partial Write for the Controller--Full Rate**



**Figure 160. Partial Write for the Controller--Half Rate**



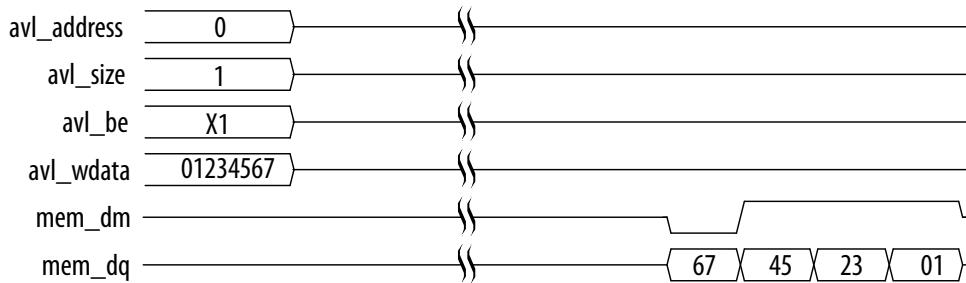
### 8.3.9.2. Partial Bursts

DIMMs that do not have the DM pins do not support partial bursts. You must write a minimum (or multiples) of memory-burst-length-equivalent words to the memory at the same time.

The following figure shows a partial burst operation for the controller.



**Figure 161. Partial Burst for Controller**



## 8.4. External Interfaces

This section discusses the interfaces between the controller and other external memory interface components.

### 8.4.1. Clock and Reset Interface

The clock and reset interface is part of the AFI interface.

The controller can have up to two clock domains, which are synchronous to each other. The controller operates with a single clock domain when there is no integrated half-rate bridge, and with two-clock domains when there is an integrated half-rate bridge. The clocks are provided by UniPHY.

The main controller clock is `afi_clk`, and the optional half-rate controller clock is `afi_half_clk`. The main and half-rate clocks must be synchronous and have a 2:1 frequency ratio. The optional quarter-rate controller clock is `afi_quarter_clk`, which must also be synchronous and have a 4:1 frequency ratio.

### 8.4.2. Avalon-ST Data Slave Interface

The Avalon-ST data slave interface consists of the following Avalon-ST channels, which together form a single data slave:

- The command channel, which serves as command and address for both read and write operations.
- The write data channel, which carries write data.
- The read data channel, which carries read data.

#### Related Information

[Avalon Interface Specifications](#)

### 8.4.3. AXI Data Slave Interface

The AXI data interface consists of the following channels, which communicate with the Avalon-ST interface through the AXI to Avalon-ST converter:



- The write address channel, which carries address information for write operations.
- The write data channel, which carries write data.
- The write response channel, which carries write response data.
- The read address channel, which carries address information for read operations.
- The read data channel, which carries read data.

#### 8.4.3.1. Enabling the AXI Interface

This section provides guidance for enabling the AXI interface.

1. To enable the AXI interface, first open in an editor the file appropriate for the required flow, as indicated below:
  - For synthesis flow: <working\_dir>/<variation\_name>/<variation\_name>.c0.v
  - For simulation flow: <working\_dir>/<variation\_name>.sim/<variation\_name>/<variation\_name>.c0.v
  - Example design fileset for synthesis: <working\_dir>/<variation\_name>\_example\_design/example\_project/<variation\_name>\_example/submodules/<variation\_name>\_example\_if0\_c0.v
  - Example design fileset for simulation: <working\_dir>/<variation\_name>\_example\_design/simulation/verilog/submodules/<variation\_name>\_example\_sim\_e0\_if0\_c0.v
2. Locate and remove the **alt\_mem\_ddrx\_mm\_st\_converter** instantiation from the .v file opened in the preceding step.
3. Instantiate the **alt\_mem\_ddrx\_axi\_st\_converter** module into the open .v file. Refer to the following code fragment as a guide:

```
module ? # ( parameter
  // AXI parameters
  AXI_ID_WIDTH = <replace parameter value>,
  AXI_ADDR_WIDTH = <replace parameter value>,
  AXI_LEN_WIDTH = <replace parameter value>,
  AXI_SIZE_WIDTH = <replace parameter value>,
  AXI_BURST_WIDTH = <replace parameter value>,
  AXI_LOCK_WIDTH = <replace parameter value>,
  AXI_CACHE_WIDTH = <replace parameter value>,
  AXI_PROT_WIDTH = <replace parameter value>,
  AXI_DATA_WIDTH = <replace parameter value>,
  AXI_RESP_WIDTH = <replace parameter value>
)
(
  // Existing ports
  ...
  // AXI Interface ports
  // Write address channel
  input wire [AXI_ID_WIDTH - 1 : 0] awid,
  input wire [AXI_ADDR_WIDTH - 1 : 0] awaddr,
  input wire [AXI_LEN_WIDTH - 1 : 0] awlen,
  input wire [AXI_SIZE_WIDTH - 1 : 0] awsize,
  input wire [AXI_BURST_WIDTH - 1 : 0] awburst,
  input wire [AXI_LOCK_WIDTH - 1 : 0] awlock,
  input wire [AXI_CACHE_WIDTH - 1 : 0] awcache,
  input wire [AXI_PROT_WIDTH - 1 : 0] awprot,
  input wire awvalid,
  output wire awready,
  // Write data channel
  input wire [AXI_ID_WIDTH - 1 : 0] wid,
```



```

input wire [AXI_DATA_WIDTH - 1 : 0] wdata,
input wire [AXI_DATA_WIDTH / 8 - 1 : 0] wstrb,
input wire wlast,
input wire wvalid,
output wire wready,
// Write response channel
output wire [AXI_ID_WIDTH - 1 : 0] bid,
output wire [AXI_RESP_WIDTH - 1 : 0] bresp,
output wire bvalid,
input wire bready,
// Read address channel
input wire [AXI_ID_WIDTH - 1 : 0] arid,
input wire [AXI_ADDR_WIDTH - 1 : 0] araddr,
input wire [AXI_LEN_WIDTH - 1 : 0] arlen,
input wire [AXI_SIZE_WIDTH - 1 : 0] arsize,
input wire [AXI_BURST_WIDTH - 1 : 0] arburst,
input wire [AXI_LOCK_WIDTH - 1 : 0] arlock,
input wire [AXI_CACHE_WIDTH - 1 : 0] arcache,
input wire [AXI_PROT_WIDTH - 1 : 0] arprot,
input wire arvalid,
output wire arready,
// Read data channel
output wire [AXI_ID_WIDTH - 1 : 0] rid,
output wire [AXI_DATA_WIDTH - 1 : 0] rdata,
output wire [AXI_RESP_WIDTH - 1 : 0] rresp,
output wire rlast,
output wire rvalid,
input wire rready
);
// Existing wire, register declaration and instantiation
...
// AXI interface instantiation
alt_mem_ddrx_axi_st_converter #
(
    .AXI_ID_WIDTH (AXI_ID_WIDTH ),
    .AXI_ADDR_WIDTH (AXI_ADDR_WIDTH ),
    .AXI_LEN_WIDTH (AXI_LEN_WIDTH ),
    .AXI_SIZE_WIDTH (AXI_SIZE_WIDTH ),
    .AXI_BURST_WIDTH (AXI_BURST_WIDTH ),
    .AXI_LOCK_WIDTH (AXI_LOCK_WIDTH ),
    .AXI_CACHE_WIDTH (AXI_CACHE_WIDTH ),
    .AXI_PROT_WIDTH (AXI_PROT_WIDTH ),
    .AXI_DATA_WIDTH (AXI_DATA_WIDTH ),
    .AXI_RESP_WIDTH (AXI_RESP_WIDTH ),
    .ST_ADDR_WIDTH (ST_ADDR_WIDTH ),
    .ST_SIZE_WIDTH (ST_SIZE_WIDTH ),
    .ST_ID_WIDTH (ST_ID_WIDTH ),
    .ST_DATA_WIDTH (ST_DATA_WIDTH ),
    .COMMAND_ARB_TYPE (COMMAND_ARB_TYPE)
)
a0
(
    .ctl_clk (afi_clk),
    .ctl_reset_n (afi_reset_n),
    .awid (awid),
    .awaddr (awaddr),
    .awlen (awlen),
    .awsize (awsize),
    .awburst (awburst),
    .awlock (awlock),
    .awcache (awcache),
    .awprot (awprot),
    .awvalid (awvalid),
    .awready (awready),
    .wid (wid),
    .wdata (wdata),
    .wstrb (wstrb),
    .wlast (wlast),
    .wvalid (wvalid),
    .wready (wready),
    .bid (bid),

```

```
.bresp (bresp),
.bvalid (bvalid),
.bready (bready),
.arid (arid),
.araddr (araddr),
.arlen (arlen),
.arsize (arsize),
.arburst (arburst),
.arlock (arlock),
.arcache (arcache),
.aprot (aprot),
.arvalid (arvalid),
.arready (arready),
.rid (rid),
.rdata (rdata),
.rresp (rresp),
.rlast (rlast),
.rvalid (rvalid),
.rready (rready),
.itf_cmd_ready (ng0_native_st_itf_cmd_ready),
.itf_cmd_valid (a0_native_st_itf_cmd_valid),
.itf_cmd (a0_native_st_itf_cmd),
.itf_cmd_address (a0_native_st_itf_cmd_address),
.itf_cmd_burstlen (a0_native_st_itf_cmd_burstlen),
.itf_cmd_id (a0_native_st_itf_cmd_id),
.itf_cmd_priority (a0_native_st_itf_cmd_priority),
.itf_cmd_autoprecharge (a0_native_st_itf_cmd_autopercharge),
.itf_cmd_multicast (a0_native_st_itf_cmd_multicast),
.itf_wr_data_ready (ng0_native_st_itf_wr_data_ready),
.itf_wr_data_valid (a0_native_st_itf_wr_data_valid),
.itf_wr_data (a0_native_st_itf_wr_data),
.itf_wr_data_byte_en (a0_native_st_itf_wr_data_byte_en),
.itf_wr_data_begin (a0_native_st_itf_wr_data_begin),
.itf_wr_data_last (a0_native_st_itf_wr_data_last),
.itf_wr_data_id (a0_native_st_itf_wr_data_id),
.itf_rd_data_ready (a0_native_st_itf_rd_data_ready),
.itf_rd_data_valid (ng0_native_st_itf_rd_data_valid),
.itf_rd_data (ng0_native_st_itf_rd_data),
.itf_rd_data_error (ng0_native_st_itf_rd_data_error),
.itf_rd_data_begin (ng0_native_st_itf_rd_data_begin),
.itf_rd_data_last (ng0_native_st_itf_rd_data_last),
.itf_rd_data_id (ng0_native_st_itf_rd_data_id)
);
```

4. Set the required parameters for the AXI interface. The following table lists the available parameters.
5. Export the AXI interface to the top-level wrapper, making it accessible to the AXI master.
6. To add the AXI interface to the Quartus Prime project:
  - On the Assignments > Settings menu in the Quartus Prime software, open the File tab.
  - Add the **alt\_mem\_ddrx\_axi\_st\_converter.v** file to the project.



### 8.4.3.2. AXI Interface Parameters

**Table 92.** AXI Interface Parameters

| Parameter Name   | Description / Value   |
|------------------|---|
| AXI_ID_WIDTH     | Width of the AXI ID bus. Default value is 4.  |
| AXI_ADDR_WIDTH   | Width of the AXI address bus. Must be set according to the Avalon interface address and data bus width as shown below: $\text{AXI\_ADDR\_WIDTH} = \text{LOCAL\_ADDR\_WIDTH} + \log_2(\text{LOCAL\_DATA\_WIDTH}/8)$<br>LOCAL_ADDR_WIDTH is the memory controller Avalon interface address width.LOCAL_DATA_WIDTH is the memory controller Avalon data interface width. |
| AXI_LEN_WIDTH    | Width of the AXI length bus. Default value is 8. Should be set to LOCAL_SIZE_WIDTH - 1, where LOCAL_SIZE_WIDTH is the memory controller Avalon interface burst size width   |
| AXI_SIZE_WIDTH   | Width of the AXI size bus. Default value is 3.  |
| AXI_BURST_WIDTH  | Width of the AXI burst bus. Default value is 2.   |
| AXI_LOCK_WIDTH   | Width of the AXI lock bus. Default value is 2.  |
| AXI_CACHE_WIDTH  | Width of the AXI cache bus. Default value is 4.   |
| AXI_PROT_WIDTH   | Width of the AXI protection bus. Default value is 3.  |
| AXI_DATA_WIDTH   | Width of the AXI data bus. Should be set to match the Avalon interface data bus width. $\text{AXI\_DATA\_WIDTH} = \text{LOCAL\_DATA\_WIDTH}$ , where LOCAL_DATA_WIDTH is the memory controller Avalon interface input data width.   |
| AXI_RESP_WIDTH   | Width of the AXI response bus. Default value is 2.  |
| ST_ADDR_WIDTH    | Width of the Avalon interface address. Must be set to match the Avalon interface address bus width. $\text{ST\_ADDR\_WIDTH} = \text{LOCAL\_ADDR\_WIDTH}$ , where LOCAL_ADDR_WIDTH is the memory controller Avalon interface address width.  |
| ST_SIZE_WIDTH    | Width of the Avalon interface burst size. $\text{ST\_SIZE\_WIDTH} = \text{AXI\_LEN\_WIDTH} + 1$   |
| ST_ID_WIDTH      | Width of the Avalon interface ID. Default value is 4. $\text{ST\_ID\_WIDTH} = \text{AXI\_ID\_WIDTH}$  |
| ST_DATA_WIDTH    | Width of the Avalon interface data. $\text{ST\_DATA\_WIDTH} = \text{AXI\_DATA\_WIDTH}$ .  |
| COMMAND_ARB_TYPE | Specifies the AXI command arbitration type, as shown:ROUND_ROBIN: arbitrates between read and write address channel in round robin fashion. Default option.WRITE_PRIORITY: write address channel has priority if both channels send request simultaneously.READ_PRIORITY: read address channel has priority if both channels send request simultaneously.             |
| REGISTERED       | Setting this parameter to 1 adds an extra register stage in the AXI interface and incurs one extra clock cycle of latency. Default value is 1.  |

### 8.4.3.3. AXI Interface Ports

**Table 93.** AXI Interface Ports

| Name    | Direction | Description  |
|---------|-----------|--|
| awid    | Input     | AXI write address channel ID bus.  |
| awaddr  | Input     | AXI write address channel address bus.   |
| awlen   | Input     | AXI write address channel length bus.  |
| awsize  | Input     | AXI write address channel size bus.  |
| awburst | Input     | AXI write address channel burst bus.(Interface supports only INCR and WRAP burst types.) |
| awlock  | Input     | AXI write address channel lock bus.(Interface does not support this feature.)            |
| awcache | Input     | AXI write address channel cache bus.(Interface does not support this feature.)           |

*continued...*



| Name    | Direction | Description  |
|---------|-----------|--|
| awprot  | Input     | AXI write address channel protection bus.(Interface does not support this feature.)  |
| awvalid | Input     | AXI write address channel valid signal.  |
| awready | Output    | AXI write address channel ready signal.  |
| wid     | Input     | AXI write address channel ID bus.  |
| wdata   | Input     | AXI write address channel data bus.  |
| wstrb   | Input     | AXI write data channel strobe bus.   |
| wlast   | Input     | AXI write data channel last burst signal.  |
| wvalid  | Input     | AXI write data channel valid signal.   |
| wready  | Output    | AXI write data channel ready signal.   |
| bid     | Output    | AXI write response channel ID bus.   |
| bresp   | Output    | AXI write response channel response bus.Response encoding information:'b00 - OKAY'b01 - Reserved'b10 - Reserved'b11 - Reserved |
| bvalid  | Output    | AXI write response channel valid signal.   |
| bready  | Input     | AXI write response channel ready signal.Must be set to 1. Interface does not support back pressure for write response channel. |
| arid    | Input     | AXI read address channel ID bus.   |
| araddr  | Input     | AXI read address channel address bus.  |
| arlen   | Input     | AXI read address channel length bus.   |
| arsize  | Input     | AXI read address channel size bus.   |
| arburst | Input     | AXI read address channel burst bus.(Interface supports only INCR and WRAP burst types.)  |
| arlock  | Input     | AXI read address channel lock bus.(Interface does not support this feature.)   |
| arcache | Input     | AXI read address channel cache bus.(Interface does not support this feature.)  |
| arprot  | Input     | AXI read address channel protection bus.(Interface does not support this feature.)   |
| arvalid | Input     | AXI read address channel valid signal.   |
| arready | Output    | AXI read address channel ready signal.   |
| rid     | Output    | AXI read data channel ID bus.  |
| rdata   | Output    | AXI read data channel data bus.  |
| rresp   | Output    | AXI read data channel response bus.Response encoding information:'b00 - OKAY'b01 - Reserved'b10 - Data error'b11 - Reserved    |
| rlast   | Output    | AXI read data channel last burst signal.   |
| rvalid  | Output    | AXI read data channel valid signal.  |
| rready  | Input     | AXI read data channel ready signal.Must be set to 1. Interface does not support back pressure for write response channel.      |

For information about the AXI specification, refer to the ARM website.

#### Related Information

[www.arm.com](http://www.arm.com)



#### 8.4.4. Controller-PHY Interface

The interface between the controller and the PHY is part of the AFI interface. The controller assumes that the PHY performs all necessary calibration processes without any interaction with the controller.

For more information about AFI signals, refer to *AFI 4.0 Specification*.

#### 8.4.5. Memory Side-Band Signals

The HPC II controller supports several optional side-band signals.

##### Self-Refresh (Low Power) Interface

The optional low power self-refresh interface consists of a request signal and an acknowledgement signal, which you can use to instruct the controller to place the memory device into self-refresh mode. This interface is clocked by afi\_clk.

When you assert the request signal, the controller places the memory device into self-refresh mode and asserts the acknowledge signal. To bring the memory device out of self-refresh mode, you deassert the request signal; the controller then deasserts the acknowledge signal when the memory device is no longer in self-refresh mode.

**Note:** For multi-rank designs using the HPC II memory controller, a self-refresh and a user-refresh cannot be made to the same memory chip simultaneously. Also, the self-refresh ack signal indicates that at least one device has entered self-refresh, but does not necessarily mean that all devices have entered self-refresh.

##### User-Controlled Refresh Interface

The optional user-controlled refresh interface consists of a request signal, a chip select signal, and an acknowledgement signal. This interface provides increased control over worst-case read latency and enables you to issue refresh bursts during idle periods. This interface is clocked by afi\_clk.

When you assert a refresh request signal to instruct the controller to perform a refresh operation, that request takes priority over any outstanding read or write requests that might be in the command queue. In addition to the request signal, you must also choose the chip to be refreshed by asserting the refresh chip select signal along with the request signal. If you do not assert the chip select signal with the request signal, unexpected behavior may result.

The controller attempts to perform a refresh as long as the refresh request signal is asserted; if you require only one refresh, you should deassert the refresh request signal after the acknowledgement signal is received. If you maintain the request signal high after the acknowledgement is sent, it would indicate that further refresh is required. You should deassert the request signal after the required number of acknowledgement/refresh is received from the controller. You can issue up to a maximum of nine consecutive refresh commands.

**Note:** For multi-rank designs using the HPC II memory controller, a self-refresh and a user-refresh cannot be made to the same memory chip simultaneously.



### Configuration and Status Register (CSR) Interface

The controller has a configuration and status register (CSR) interface that allows you to configure timing parameters, address widths, and the behavior of the controller. The CSR interface is a 32-bit Avalon-MM slave of fixed address width; if you do not need this feature, you can disable it to save area.

This interface is clocked by `csr_clk`, which is the same as `afi_clk`, and is always synchronous relative to the main data slave interface.

### 8.4.6. Controller External Interfaces

The following table lists the controller's external interfaces.

**Table 94. Summary of Controller External Interfaces**

| Interface Name  | Display Name                                | Type   | Description   |
|---|---|--|---|
| <b>Clock and Reset Interface</b>  |   |  |   |
| Clock and Reset Interface   | Clock and Reset Interface                   | AFI <sup>(1)</sup>                               | Clock and reset generated by UniPHY to the controller.  |
| <b>Avalon-ST Data Slave Interface</b>   |   |  |   |
| Command Channel   | Avalon-ST Data Slave Interface              | Avalon-ST <sup>(2)</sup>                         | Address and command channel for read and write, single command single data (SCSD).                      |
| Write Data Channel  | Avalon-ST Data Slave Interface              | Avalon-ST <sup>(2)</sup>                         | Write Data Channel, single command multiple data (SCMD).  |
| Read Data Channel   | Avalon-ST Data Slave Interface              | Avalon-ST <sup>(2)</sup>                         | Read data channel, SCMD with read data error response.  |
| <b>Controller-PHY Interface</b>   |   |  |   |
| AFI 4.0   | AFI Interface                               | AFI <sup>(1)</sup>                               | Interface between controller and PHY.   |
| <b>Memory Side-Band Signals</b>   |   |  |   |
| Self Refresh (Low Power) Interface  | Self Refresh (Low Power) Interface          | Avalon Control & Status Interface <sup>(2)</sup> | SDRAM-specific signals to place memory into low-power mode.   |
| User-Controller Refresh Interface   | User-Controller Refresh Interface           | Avalon Control & Status Interface <sup>(2)</sup> | SDRAM-specific signals to request memory refresh.   |
| <b>Configuration and Status Register (CSR) Interface</b>  |   |  |   |
| CSR   | Configuration and Status Register Interface | Avalon-MM <sup>(2)</sup>                         | Enables on-the-fly configuration of memory timing parameters, address widths, and controller behaviour. |
| Notes:  |   |  |   |
| 1. For information about AFI signals, refer to <i>AFI 4.0 Specification</i> in the <i>Functional Description - UniPHY chapter</i> . |   |  |   |
| 2. For information about Avalon signals, refer to <i>Avalon Interface Specifications</i> .  |   |  |   |

#### Related Information

- [Avalon Interface Specifications](#)
- [Functional Description—UniPHY](#) on page 13



## 8.5. Top-Level Signals Description

The top-level signals include clock and reset signals, local interface signals, controller interface signals, and CSR interface signals.

### 8.5.1. Clock and Reset Signals

The following table lists the clock and reset signals.

**Note:** The suffix `_n` denotes active low signals.

**Table 95. Clock and Reset Signals**

| Name                     | Direction               | Description  |
|--------------------------|-------------------------|--|
| global_reset_n           | Input                   | The asynchronous reset input to the controller. The IP core derives all other reset signals from resynchronized versions of this signal. This signal holds the PHY, including the PLL, in reset while low.   |
| pll_ref_clk              | Input                   | The reference clock input to PLL.  |
| phy_clk                  | Output                  | The system clock that the PHY provides to the user. All user inputs to and outputs from the controller must be synchronous to this clock.  |
| reset_phy_clk_n          | Output                  | The reset signal that the PHY provides to the user. The IP core asserts <code>reset_phy_clk_n</code> asynchronously and deasserts synchronously to <code>phy_clk</code> clock domain.  |
| aux_full_rate_clk        | Output                  | An alternative clock that the PHY provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate designs, this clock is twice the frequency of the <code>phy_clk</code> and you can use it whenever you require a 2x clock. In full-rate designs, the same PLL output as the <code>phy_clk</code> signal drives this clock.   |
| aux_half_rate_clk        | Output                  | An alternative clock that the PHY provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate designs, this clock is half the frequency of the <code>phy_clk</code> and you can use it, for example to clock the user side of a half-rate bridge. In half-rate designs, or if the <b>Enable Half Rate Bridge</b> option is turned on. The same PLL output that drives the <code>phy_clk</code> signal drives this clock. |
| dll_reference_clk        | Output                  | Reference clock to feed to an externally instantiated DLL.   |
| reset_request_n          | Output                  | Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using Intel advises you detect a reset request on a falling edge rather than by level detection.   |
| soft_reset_n             | Input                   | Edge detect reset input for control by other system reset logic. Assert to cause a complete reset to the PHY, but not to the PLL that the PHY uses.  |
| seriesterminationcontrol | Input (for OCT slave)   | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block ( <code>alt_oct</code> ) or another UniPHY instance that is set to OCT master mode.  |
|                          | Output (for OCT master) | Unconnected PHY signal, available for sharing with another PHY.  |

*continued...*



| Name   | Direction               | Description  |
|--|-------------------------|--|
| parallelterminationcontrol   | Input (for OCT slave)   | Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode. |
|  | Output (for OCT master) | Unconnected PHY signal, available for sharing with another PHY.  |
| oct_rdn  | Input (for OCT master)  | Must connect to calibration resistor tied to GND on the appropriate RDN pin on the device. (Refer to appropriate device handbook.)   |
| oct_rup  | Input (for OCT master)  | Must connect to calibration resistor tied to Vccio on the appropriate RUP pin on the device. (See appropriate device handbook.)  |
| dqs_delay_ctrl_import  | Input                   | Allows the use of DLL in another PHY instance in this PHY instance. Connect the <code>export</code> port on the PHY instance with a DLL to the <code>import</code> port on the other PHY instance.       |
| csr_clk  | Output                  | Clock for the configuration and status register (CSR) interface, which is the same as <code>afi_clk</code> and is always synchronous relative to the main data slave interface.                          |
| <p>Note:</p> <ol style="list-style-type: none"><li>Applies only to the hard memory controller with multiport front end available in Arria V and Cyclone V devices.</li></ol> |                         |  |

## 8.5.2. Local Interface Signals

The following table lists the controller local interface signals.

**Table 96. Local Interface Signals**

| Signal Name               | Direction | Description   |
|---------------------------|-----------|---|
| avl_addr[] <sup>(1)</sup> | Input     | <p>Memory address at which the burst should start. By default, the IP core maps local address to the bank interleaving scheme. You can change the ordering via the <b>Local-to-Memory Address Mapping</b> option in the <b>Controller Settings</b> page.</p> <p>This signal must remain stable only during the first transaction of a burst. The <code>constantBurstBehavior</code> property is always false for UniPHY controllers.</p> <p>The IP core sizes the width of this bus according to the following equations:</p> <ul style="list-style-type: none"><li>• Full rate controllers:<br/>For one chip select: <math>\text{width} = \text{row bits} + \text{bank bits} + \text{column bits} - 1</math><br/>For multiple chip selects: <math>\text{width} = \text{chip bits} * \text{row bits} + \text{bank bits} + \text{column bits} - 1</math></li><li>• Half rate controllers:<br/>For one chip select: <math>\text{width} = \text{row bits} + \text{bank bits} + \text{column bits} - 2</math><br/>For multiple chip selects: <math>\text{width} = \text{chip bits} * \text{row bits} + \text{bank bits} + \text{column bits} - 2</math></li></ul> <p>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 24 bits wide. To map <code>local_address</code> to bank, row and column address:</p> <div style="background-color: #f0f0f0; padding: 5px;"><code>avl_addr is 24 bits wide<br/>avl_addr[23:11] = row address[12:0]<br/>avl_addr[10:9] = bank address [1:0]<br/>avl_addr[8:0] = column address[9:1]</code></div> <p>The IP core ignores the least significant bit (LSB) of the column address (multiples of two) on the memory side, because the local data width is twice that of the memory data bus width.</p> |

*continued...*



| Signal Name        | Direction | Description  |
|--------------------|-----------|--|
|                    |           | <p>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 23 bits wide. To map local_address to bank, row and column address:</p> <pre>avl_addr is 23 bits wide avl_addr[22:10] = row address[12:0] avl_addr[9:8] = bank address [1:0] avl_addr[7:0] = column address[9:2]</pre> <p>The IP core ignores two LSBs of the column address (multiples of four) on the memory side, because the local data width is four times that of the memory data bus width.</p> <ul style="list-style-type: none"> <li>Quarter rate controllers:<br/>For one chip select: width = row bits + bank bits + column bits – 3<br/>For multiple chip selects: width = chip bits* + row bits + bank bits + column bits – 3</li> <li>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 22 bits wide.<br/>(* <i>chip bits</i> is a derived value indicating the number of address bits necessary to uniquely address every memory rank in the system; this value is not user configurable.)</li> <li>Full-rate hard memory controllers (Arria V and Cyclone V):<br/>For one chip select: width = row bits + bank bits + column bits - log2(local avalon data width/memory DQ width)<br/>For multiple chip selects: width = chip bits* + row bits + bank bits + column bits - log2(local avalon data width/memory data width)</li> </ul> <p>If the local Avalon data width is 32, the memory DQ width is 8, the bank address is 3 bits wide, the row is 12 bits wide and the column is 8 bits wide, the local address is 21 bits wide. To map local_address to bank, row and column address:</p> <p>avl_addr is 21 bits wide</p> <pre>avl_addr[20:9] = row address[11:0] avl_addr[8:6] = bank address [2:0] avl_addr[5:0] = column address[7:2]</pre> <p>The IP core ignores the two least significant bits of the column address on the memory side because the local data width is four times that of the memory data bus width (Multi-Port Frontend).</p> |
| avl_be[ ] (2)      | Input     | <p>Byte enable signal, which you use to mask off individual bytes during writes. avl_be is active high; mem_dm is active low.</p> <p>To map avl_wdata and avl_be to mem_dq and mem_dm, consider a full-rate design with 32-bit avl_wdata and 16-bit mem_dq.</p> <pre>avl_wdata = &lt; 22334455 &gt;&lt; 667788AA &gt;&lt; BBCCDDEE&gt; avl_be = &lt; 1100 &gt;&lt; 0110 &gt;&lt; 1010 &gt;</pre> <p>These values map to:</p> <pre>Mem_dq = &lt;4455&gt;&lt;2233&gt;&lt;88AA&gt;&lt;6677&gt;&lt;DDEE&gt;&lt;BBCC&gt; Mem_dm = &lt;1 1 &gt;&lt;0 0 &gt;&lt;0 1 &gt;&lt;1 0 &gt;&lt;0 1 &gt;&lt;0 1 &gt;</pre>  |
| avl_burstbegin (3) | Input     | <p>The Avalon burst begin strobe, which indicates the beginning of an Avalon burst. Unlike all other Avalon-MM signals, the burst begin signal is not dependant on avl_ready.</p> <p>For write transactions, assert this signal at the beginning of each burst transfer and keep this signal high for one cycle per burst transfer, even if the slave deasserts avl_ready. The IP core samples this signal at the rising edge of phy_clk when avl_write_req is asserted. After the slave deasserts the avl_ready signal, the master keeps all the write request signals asserted until avl_ready signal becomes high again.</p>  |

*continued...*



| Signal Name                                   | Direction | Description  |
|---|-----------|--|
|   |           | For read transactions, assert this signal for one clock cycle when read request is asserted and <code>avl_addr</code> from which the data should be read is given to the memory. After the slave deasserts <code>avl_ready</code> ( <code>waitrequest_n</code> in Avalon interface), the master keeps all the read request signals asserted until <code>avl_ready</code> becomes high again.   |
| <code>avl_read_req</code> <sup>(4)</sup>      | Input     | Read request signal. You cannot assert read request and write request signals at the same time. The controller must deassert <code>reset_phy_clk_n</code> before you can assert <code>avl_read_req</code> .  |
| <code>local_refresh_req</code>                | Input     | User-controlled refresh request. If <b>Enable User Auto-Refresh Controls</b> option is turned on, <code>local_refresh_req</code> becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including grouping together multiple refresh commands. Refresh requests take priority over read and write requests, unless the IP core is already processing the requests.  |
| <code>local_refresh_chip</code>               | Input     | Controls which chip to issue the user refresh to. The IP core uses this active high signal with <code>local_refresh_req</code> . This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip.<br>For example: If <code>local_refresh_chip</code> signal is assigned with a value of <code>4'b0101</code> , the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.   |
| <code>avl_size[]</code> <sup>(5)</sup>        | Input     | Controls the number of beats in the requested read or write access to memory, encoded as a binary number. In UniPHY, the IP core supports Avalon burst lengths from 1 to 1024. The IP core derives the width of this signal based on the burst count that you specify in the <b>Maximum Avalon-MM burst length</b> option. With the derived width, you specify a value ranging from 1 to the local maximum burst count specified.<br>This signal must remain stable only during the first transaction of a burst. The <code>constantBurstBehavior</code> property is always false for UniPHY controllers.  |
| <code>avl_wdata[]</code> <sup>(6)</sup>       | Input     | Write data bus. The width of <code>avl_wdata</code> is twice that of the memory data bus for a full-rate controller, four times the memory data bus for a half-rate controller, and eight times the memory data bus for a quarter-rate controller. If <b>Generate power-of-2 data bus widths for Qsys and SOPC Builder</b> is enabled, the width is rounded down to the nearest power of 2.  |
| <code>avl_write_req</code> <sup>(7)</sup>     | Input     | Write request signal. You cannot assert read request and write request signal at the same time. The controller must deassert <code>reset_phy_clk_n</code> before you can assert <code>avl_write_req</code> .   |
| <code>local_autopch_req</code> <sup>(8)</sup> | Input     | User control of autoprecharge. If you turn on <b>Enable Auto- Precharge Control</b> , the <code>local_autopch_req</code> signal becomes available and you can request the controller to issue an autoprecharge write or autoprecharge read command.<br>These commands cause the memory to issue a precharge command to the current bank at the appropriate time without an explicit precharge command from the controller. This feature is particularly useful if you know the current read or write is the last one you intend to issue to the currently open row. The next time you need to use that bank, the access could be quicker as the controller does not need to precharge the bank before activating the row you wish to access.<br>Upon receipt of the <code>local_autopch_req</code> signal, the controller evaluates the pending commands in the command buffer and determines the most efficient autoprecharge operation to perform, reordering commands if necessary.<br>The controller must deassert <code>reset_phy_clk_n</code> before you can assert <code>local_autopch_req</code> . |

*continued...*



| Signal Name                     | Direction | Description   |
|---------------------------------|-----------|---|
| local_self_rfsh_chip            | Input     | <p>Controls which chip to issue the user refresh to. The IP core uses this active high signal with local_self_rfsh_req. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip.</p> <p>For example: If local_self_rfsh_chip signal is assigned with a value of 4'b0101, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.</p>  |
| local_self_rfsh_req             | Input     | <p>User control of the self-refresh feature. If you turn on <b>Enable Self-Refresh Controls</b>, you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting local_self_rfsh_ack. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting local_self_rfsh_req and the controller responds by deasserting local_self_rfsh_ack when it has successfully brought the memory out of the self-refresh state.</p>   |
| local_init_done                 | Output    | <p>When the memory initialization, training, and calibration are complete, the PHY sequencer asserts ctrl_usr_mode_rdy to the memory controller, which then asserts this signal to indicate that the memory interface is ready for use.</p>   |
| local_cal_success               | Output    | <p>When the memory initialization, training, and calibration completes successfully, the controller asserts this signal coincident with local_init_done to indicate the memory interface is ready for use.</p>  |
| local_cal_fail                  | Output    | <p>When the memory initialization, training, or calibration fails, the controller asserts this signal to indicate that calibration failed. The local_init_done signal will not assert when local_cal_fail asserts.</p>  |
| avl_rdata[] <sup>(9)</sup>      | Output    | <p>Read data bus. The width of avl_rdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. If <b>Generate power-of-2 data bus widths for Qsys and SOPC Builder</b> is enabled, the width is rounded down to the nearest power of 2.</p>   |
| avl_rdata_error <sup>(10)</sup> | Output    | <p>Asserted if the current read data has an error. This signal is only available if you turn on <b>Enable Error Detection and Correction Logic</b>. The controller asserts this signal with the avl_rdata_valid signal.</p> <p>If the controller encounters double-bit errors, no correction is made and the controller asserts this signal.</p>  |
| avl_rdata_valid <sup>(11)</sup> | Output    | <p>Read data valid signal. The avl_rdata_valid signal indicates that valid data is present on the read data bus.</p>  |
| avl_ready <sup>(12)</sup>       | Output    | <p>The avl_ready signal indicates that the controller is ready to accept request signals. If controller asserts the avl_ready signal in the clock cycle that it asserts a read or write request, the controller accepts that request. The controller deasserts the avl_ready signal to indicate that it cannot accept any more requests. The controller can buffer eight read or write requests, after which the avl_ready signal goes low.</p> <p>The avl_ready signal is deasserted when any of the following are true:</p> <ul style="list-style-type: none"> <li>• The Timing bank Pool is full.</li> <li>• The FIFO register that stores read data from the memory device is full.</li> <li>• The write data FIFO register is full.</li> <li>• The controller is waiting for write data when in ECC mode.</li> </ul> |

*continued...*



| Signal Name                   | Direction | Description   |
|-------------------------------|-----------|---|
| local_refresh_ack             | Output    | Refresh request acknowledge, which the controller asserts for one clock cycle every time it issues a refresh. Even if you do not turn on <b>Enable User Auto-Refresh Controls</b> , local_refresh_ack still indicates to the local interface that the controller has just issued a refresh command. |
| local_self_rfsh_ack           | Output    | Self refresh request acknowledge signal. The controller asserts and deasserts this signal in response to the local_self_rfsh_req signal.  |
| local_power_down_ack          | Output    | Auto power-down acknowledge signal. The controller asserts this signal for one clock cycle every time auto power-down is issued.  |
| ecc_interrupt <sup>(13)</sup> | Output    | Interrupt signal from the ECC logic. The controller asserts this signal when the ECC feature is turned on, and the controller detects an error.   |

Notes to Table:

1. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_addr becomes a per port value, avl\_addr\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
2. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_be becomes a per port value, avl\_be\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
3. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_burstbegin becomes a per port value, avl\_burstbegin\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
4. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_read\_req becomes a per port value, avl\_read\_req\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
5. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_size becomes a per port value, avl\_size\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
6. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_wdata becomes a per port value, avl\_wdata\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
7. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_write\_req becomes a per port value, avl\_write\_req\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
8. This signal is not applicable to the hard memory controller.
9. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_rdata becomes a per port value, avl\_rdata\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
10. This signal is not applicable to the hard memory controller.
11. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_rdata\_valid becomes a per port value, avl\_rdata\_valid\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
12. For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, avl\_ready becomes a per port value, avl\_ready\_#, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
13. This signal is not applicable to the hard memory controller.

### 8.5.3. Controller Interface Signals

The following table lists the controller interface signals.

**Table 97. Interface Signals**

| Signal Name | Direction     | Description   |
|-------------|---------------|---|
| mem_dq[ ]   | Bidirectional | Memory data bus. This bus is half the width of the local read and write data busses.                              |
| mem_dqs[ ]  | Bidirectional | Memory data strobe signal, which writes data into the memory device and captures read data into the Intel device. |

*continued...*



| Signal Name   | Direction     | Description   |
|---|---------------|---|
| mem_dqs_n[ ]  | Bidirectional | Inverted memory data strobe signal, which with the mem_dqs signal improves signal integrity.  |
| mem_ck  | Output        | Clock for the memory device.  |
| mem_ck_n  | Output        | Inverted clock for the memory device.   |
| mem_addr[ ]   | Output        | Memory address bus.   |
| mem_ac_parity <sup>(1)</sup>  | Output        | Address or command parity signal generated by the PHY and sent to the DIMM. DDR3 SDRAM only.  |
| mem_ba[ ]   | Output        | Memory bank address bus.  |
| mem_cas_n   | Output        | Memory column address strobe signal.  |
| mem_cke[ ]  | Output        | Memory clock enable signals.  |
| mem_cs_n[ ]   | Output        | Memory chip select signals.   |
| mem_dm[ ]   | Output        | Memory data mask signal, which masks individual bytes during writes.  |
| mem_odi   | Output        | Memory on-die termination control signal.   |
| mem_ras_n   | Output        | Memory row address strobe signal.   |
| mem_we_n  | Output        | Memory write enable signal.   |
| parity_error_n <sup>(1)</sup>   | Output        | This signal is not used and should be ignored. The PHY does not have capture circuitry to trigger on the falling edge of mem_error_out_n. See below for a complete description of mem_error_out_n, and recommendations.   |
| mem_err_out_n <sup>(1)</sup>  | Input         | This is an output of registered DIMMs. When an address-and-command parity error is detected, DDR3 registered DIMMs assert the mem_err_out_n signal in accordance with the memory buffer configuration. Unlike ECC on the data bus, the controller does not automatically correct errors on the address-and-command bus. You should connect this pin to your own falling edge detection circuitry in order to capture when a parity error occurs. Upon error detection, action may be taken such as causing a system interrupt, or an appropriate event. |
| <p>Note:</p> <ol style="list-style-type: none"> <li>1. This signal is for registered DIMMs only.</li> </ol> |               |   |

### 8.5.4. CSR Interface Signals

The following table lists the CSR interface signals.

**Table 98. CSR Interface Signals**

| Signal Name | Direction | Description   |
|-------------|-----------|---|
| csr_addr[ ] | Input     | Register map address. The width of csr_addr is 16 bits. |

*continued...*



| Signal Name  | Direction | Description  |
|--|-----------|--|
| csr_be[]   | Input     | Byte-enable signal, which you use to mask off individual bytes during writes. csr_be is active high.   |
| csr_clk <sup>(1)</sup>   | Output    | Clock for the configuration and status register (CSR) interface, which is the same as afi_clk and is always synchronous relative to the main data slave interface.   |
| csr_wdata[]  | Input     | Write data bus. The width of csr_wdata is 32 bits.   |
| csr_write_req  | Input     | Write request signal. You cannot assert csr_write_req and csr_read_req signals at the same time.   |
| csr_read_req   | Input     | Read request signal. You cannot assert csr_read_req and csr_write_req signals at the same time.  |
| csr_rdata[]  | Output    | Read data bus. The width of csr_rdata is 32 bits.  |
| csr_rdata_valid  | Output    | Read data valid signal. The csr_rdata_valid signal indicates that valid data is present on the read data bus.  |
| csr_waitrequest  | Output    | The csr_waitrequest signal indicates that the HPC II is busy and not ready to accept request signals. If the csr_waitrequest signal goes high in the clock cycle when a read or write request is asserted, that request is not accepted. If the csr_waitrequest signal goes low, the HPC II is then ready to accept more requests. |
| Note to Table:<br>1. Applies only to the hard memory controller with multiport front end available in Arria V and Cyclone V devices. |           |  |

### 8.5.5. Soft Controller Register Map

The soft controller register map allows you to control the soft memory controller settings.

*Note:* Dynamic reconfiguration is not currently supported.

The following table lists the register map for the controller.

**Table 99. Soft Controller Register Map**

| Address | Bit   | Name           | Default | Access     | Description   |
|---------|-------|----------------|---------|------------|---|
| 0x100   | 0     | Reserved.      | 0       | —          | Reserved for future use.  |
|         | 1     | Reserved.      | 0       | —          | Reserved for future use.  |
|         | 2     | Reserved.      | 0       | —          | Reserved for future use.  |
|         | 7:3   | Reserved.      | 0       | —          | Reserved for future use.  |
|         | 13:8  | Reserved.      | 0       | —          | Reserved for future use.  |
|         | 30:14 | Reserved.      | 0       | —          | Reserved for future use.  |
| 0x110   | 15:0  | AUTO_PD_CYCLES | 0x0     | Read write | The number of idle clock cycles after which the controller should place the memory into power-down mode. The controller is considered to be idle if there are no commands in the command queue. Setting this register to 0 disables the auto power-down |

*continued...*



| Address | Bit   | Name                             | Default | Access     | Description   |
|---------|-------|----------------------------------|---------|------------|---|
|         |       |                                  |         |            | mode. The default value of this register depends on the values set during the generation of the design.   |
|         | 16    | Reserved.                        | 0       | —          | Reserved for future use.  |
|         | 17    | Reserved.                        | 0       | —          | Reserved for future use.  |
|         | 18    | Reserved.                        | 0       | —          | Reserved for future use.  |
|         | 19    | Reserved.                        | 0       | —          | Reserved for future use.  |
|         | 21:20 | ADDR_ORDER                       | 00      | Read write | 00 - Chip, row, bank, column.01 - Chip, bank, row, column.10 - reserved for future use.11 - Reserved for future use.  |
|         | 22    | Reserved.                        | 0       | —          | Reserved for future use.  |
|         | 24:23 | Reserved.                        | 0       | —          | Reserved for future use.  |
|         | 30:24 | Reserved                         | 0       | —          | Reserved for future use.  |
| 0x120   | 7:0   | Column address width             | —       | Read write | The number of column address bits for the memory devices in your memory interface. The range of legal values is 7-12.   |
| 0x120   | 15:8  | Row address width                | —       | Read write | The number of row address bits for the memory devices in your memory interface. The range of legal values is 12-16.   |
| 0x120   | 19:16 | Bank address width               | —       | Read write | The number of bank address bits for the memory devices in your memory interface. The range of legal values is 2-3.  |
| 0x120   | 23:20 | Chip select address width        | —       | Read write | The number of chip select address bits for the memory devices in your memory interface. The range of legal values is 0-2. If there is only one single chip select in the memory interface, set this bit to 0. |
| 0x120   | 31:24 | Reserved.                        | 0       | —          | Reserved for future use.  |
| 0x121   | 31:0  | Data width representation (word) | —       | Read only  | The number of DQS bits in the memory interface. This bit can be used to derive the width of the memory interface by multiplying this value by the number of DQ pins per DQS pin (typically 8).                |
| 0x122   | 7:0   | Chip select representation       | —       | Read only  | The number of chip select in binary representation. For example, a design with 2 chip selects has the value of 00000011.  |
| 0x122   | 31:8  | Reserved.                        | 0       | —          | Reserved for future use.  |
| 0x123   | 3:0   | tRCD                             | —       | Read write | The activate to read or write a timing parameter. The range of legal values is 2-11 cycles.   |

*continued...*



| Address | Bit   | Name             | Default | Access     | Description  |
|---------|-------|------------------|---------|------------|--|
|         | 7:4   | tRRD             | —       | Read write | The activate to activate a timing parameter. The range of legal values is 2-8 cycles.  |
|         | 11:8  | tRP              | —       | Read write | The precharge to activate a timing parameter. The range of legal values is 2-11 cycles.  |
|         | 15:12 | tMRD             | —       | Read write | The mode register load time parameter. This value is not used by the controller, as the controller derives the correct value from the memory type setting. |
|         | 23:16 | tRAS             | —       | Read write | The activate to precharge a timing parameter. The range of legal values is 4-29 cycles.  |
|         | 31:24 | tRC              | —       | Read write | The activate to activate a timing parameter. The range of legal values is 8-40 cycles.   |
| 0x124   | 3:0   | tWTR             | —       | Read write | The write to read a timing parameter. The range of legal values is 1-10 cycles.  |
|         | 7:4   | tRTP             | —       | Read write | The read to precharge a timing parameter. The range of legal values is 2-8 cycles.   |
|         | 15:8  | tFAW             | —       | Read write | The four-activate window timing parameter. The range of legal values is 6-32 cycles.   |
|         | 31:16 | Reserved.        | 0       | —          | Reserved for future use.   |
| 0x125   | 15:0  | tREFI            | —       | Read write | The refresh interval timing parameter. The range of legal values is 780-6240 cycles.   |
|         | 23:16 | tRFC             | —       | Read write | The refresh cycle timing parameter. The range of legal values is 12-255 cycles.  |
|         | 31:24 | Reserved.        | 0       | —          | Reserved for future use.   |
| 0x126   | 3:0   | Reserved.        | 0       | —          | Reserved for future use.   |
|         | 7:4   | Reserved.        | 0       | —          | Reserved for future use.   |
|         | 11:8  | Reserved.        | 0       | —          | Reserved for future use.   |
|         | 15:12 | Reserved.        | 0       | —          | Reserved for future use.   |
|         | 23:16 | Burst Length     | —       | Read write | Value must match memory burst length.  |
|         | 31:24 | Reserved.        | 0       | —          | Reserved for future use.   |
| 0x130   | 0     | ENABLE_ECC       | 1       | Read write | When this bit equals 1, it enables the generation and checking of ECC. This bit is only active if ECC was enabled during IP parameterization.              |
|         | 1     | ENABLE_AUTO_CORR | —       | Read write | When this bit equals 1, it enables auto-correction when a single-bit error is detected.  |

*continued...*



| Address | Bit   | Name              | Default | Access     | Description  |
|---------|-------|-------------------|---------|------------|--|
|         | 2     | GEN_SBE           | 0       | Read write | When this bit equals 1, it enables the deliberate insertion of single-bit errors, bit 0, in the data written to memory. This bit is used only for testing purposes.        |
|         | 3     | GEN_DBE           | 0       | Read write | When this bit equals 1, it enables the deliberate insertion of double-bit errors, bits 0 and 1, in the data written to memory. This bit is used only for testing purposes. |
|         | 4     | ENABLE_INTR       | 1       | Read write | When this bit equals 1, it enables the interrupt output.   |
|         | 5     | MASK_SBE_INTR     | 0       | Read write | When this bit equals 1, it masks the single-bit error interrupt.   |
|         | 6     | MASK_DBE_INTR     | 0       | Read write | When this bit equals 1, it masks the double-bit error interrupt  |
|         | 7     | CLEAR             | 0       | Read write | When this bit equals 1, writing to this self-clearing bit clears the interrupt signal, and the error status and error address registers.                                   |
|         | 8     | MASK_CORDROP_INTR | 0       | Read write | When this bit equals 1, the dropped autocorrection error interrupt is dropped.   |
|         | 9     | Reserved.         | 0       | —          | Reserved for future use.   |
|         | 0x131 | SBE_ERROR         | 0       | Read only  | Set to 1 when any single-bit errors occur.   |
|         | 1     | DBE_ERROR         | 0       | Read only  | Set to 1 when any double-bit errors occur.   |
|         | 2     | CORDROP_ERROR     | 0       | Read only  | Value is set to 1 when any controller-scheduled autocorrections are dropped.   |
|         | 7:3   | Reserved.         | 0       | —          | Reserved for future use.   |
|         | 15:8  | SBE_COUNT         | 0       | Read only  | Reports the number of single-bit errors that have occurred since the status register counters were last cleared.   |
|         | 23:16 | DBE_COUNT         | 0       | Read only  | Reports the number of double-bit errors that have occurred since the status register counters were last cleared.   |
|         | 31:24 | CORDROP_COUNT     | 0       | Read only  | Reports the number of controller-scheduled autocorrections dropped since the status register counters were last cleared.   |
|         | 0x132 | ERR_ADDR          | 0       | Read only  | The address of the most recent ECC error. This address is a memory burst-aligned local address.  |

*continued...*



| Address | Bit   | Name         | Default | Access     | Description   |
|---------|-------|--------------|---------|------------|---|
| 0x133   | 31:0  | CORDROP_ADDR | 0       | Read only  | The address of the most recent autocorrection that was dropped. This is a memory burst-aligned local address. |
| 0x134   | 0     | REORDER_DATA | —       | Read write |   |
|         | 15:1  | Reserved.    | 0       | —          | Reserved for future use.  |
|         | 23:16 | STARVE_LIMIT | 0       | Read write | Number of commands that can be served before a starved command.   |
|         | 31:24 | Reserved.    | 0       | —          | Reserved for future use.  |

### 8.5.6. Hard Controller Register Map

The hard controller register map allows you to control the hard memory controller settings.

*Note:* Dynamic reconfiguration is not currently supported.

The following table lists the register map for the hard controller.

**Table 100. Hard Controller Register Map**

| Address | Bit  | Name           | Default | Access     | Description  |
|---------|------|----------------|---------|------------|--|
| 0x000   | 3:0  | CFG_CAS_WR_LAT | 0       | Read/Write | Memory write latency.                                      |
| 0x001   | 4:0  | CFG_ADD_LAT    | 0       | Read/Write | Memory additive latency.                                   |
| 0x002   | 4:0  | CFG_TCL        | 0       | Read/Write | Memory read latency.                                       |
| 0x003   | 3:0  | CFG_TRRD       | 0       | Read/Write | The activate to activate different banks timing parameter. |
| 0x004   | 5:0  | CFG_TFAW       | 0       | Read/Write | The four-activate window timing parameter.                 |
| 0x005   | 7:0  | CFG_TRFC       | 0       | Read/Write | The refresh cycle timing parameter.                        |
| 0x006   | 12:0 | CFG_TREFI      | 0       | Read/Write | The refresh interval timing parameter.                     |
| 0x008   | 3:0  | CFG_TREFI      | 0       | Read/Write | The activate to read/write timing parameter.               |
| 0x009   | 3:0  | CFG_TRP        | 0       | Read/Write | The precharge to activate timing parameter.                |
| 0x00A   | 3:0  | CFG_TWR        | 0       | Read/Write | The write recovery timing.                                 |
| 0x00B   | 3:0  | CFG_TWTR       | 0       | Read/Write | The write to read timing parameter.                        |
| 0x00C   | 3:0  | CFG_TRTP       | 0       | Read/Write | The read to precharge timing parameter.                    |
| 0x00D   | 4:0  | CFG_TRAS       | 0       | Read/Write | The activate to precharge timing parameter.                |

*continued...*



| Address | Bit  | Name                      | Default | Access     | Description  |
|---------|------|---------------------------|---------|------------|--|
| 0x00E   | 5:0  | CFG_TRC                   | 0       | Read/Write | The activate to activate timing parameter.   |
| 0x00F   | 15:0 | CFG_AUTO_PD_CYCLES        | 0       | Read/Write | The number of idle clock cycles after which the controller should place the memory into power-down mode.   |
| 0x011   | 9:0  | CFG_SELF_RFSH_EXIT_CYCLES | 0       | Read/Write | The self-refresh exit cycles.  |
| 0x013   | 9:0  | CFG_PDN_EXIT_CYCLES       | 0       | Read/Write | The power down exit cycles.  |
| 0x015   | 3:0  | CFG_TMRD                  | 0       | Read/Write | Mode register timing parameter.  |
| 0x016   | 4:0  | CFG_COL_ADDR_WIDTH        | 0       | Read/Write | The number of column address bits for the memory devices in your memory interface.   |
| 0x017   | 4:0  | CFG_ROW_ADDR_WIDTH        | 0       | Read/Write | The number of row address bits for the memory devices in your memory interface.  |
| 0x018   | 2:0  | CFG_BANK_ADDR_WIDTH       | 0       | Read/Write | The number of bank address bits for the memory devices in your memory interface.   |
| 0x019   | 2:0  | CFG_CS_ADDR_WIDTH         | 0       | Read/Write | The number of chip select address bits for the memory devices in your memory interface.  |
| 0x035   | 3:0  | CFG_TCCD                  | 0       | Read/Write | CAS#-to-CAS# command delay.  |
| 0x035   | 3:0  | CFG_WRITE_ODT_CHIP        | 0       | Read/Write | CAS#-to-CAS# command delay.  |
| 0x037   | 3:0  | CFG_READ_ODT_CHIP         | 0       | Read/Write | Read ODT Control.  |
| 0x040   | 2:0  | CFG_TYPE                  | 0       | Read/Write | Selects memory type.   |
|         | 7:3  | CFG_BURST_LENGTH          | 0       | Read/Write | Configures burst length as a static decimal value.   |
|         | 9:8  | CFG_ADDR_ORDER            | 0       | Read/Write | Address order selection.   |
|         | 10   | CFG_ENABLE_ECC            | 0       | Read/Write | Enable the generation and checking of ECC.   |
|         | 11   | CFG_ENABLE_AUTO_CORR      | 0       | Read/Write | Enable auto correction when single bit error is detected.  |
|         | 12   | CFG_GEN_SBE               | 0       | Read/Write | When this bit equals 1, it enables the deliberate insertion of single-bit errors, bit 0, in the data written to memory. This bit is used only for testing purposes.        |
|         | 13   | CFG_GEN_DBE               | 0       | Read/Write | When this bit equals 1, it enables the deliberate insertion of double-bit errors, bits 0 and 1, in the data written to memory. This bit is used only for testing purposes. |
|         | 14   | CFG_REORDER_DATA          | 0       | Read/Write | Enable Data Reordering.  |
|         | 15   | CFG_USER_RFSH             | 0       | Read/Write | Enable User Refresh.   |

*continued...*



| Address | Bit  | Name                            | Default | Access      | Description   |
|---------|------|---------------------------------|---------|-------------|---|
|         | 16   | CFG_REGDIMM_ENABLE              | 0       | Read/Write  | REG DIMM Configuration.   |
|         | 17   | CFG_ENABLE_DQS_TRACKING         | 0       | Read/Write  | Enable DQS Tracking.  |
|         | 18   | CFG_OUTPUT_REGD                 | 0       | Read/Write  | Enable Registered Output.   |
|         | 19   | CFG_ENABLE_NO_DM                | 0       | Read/Write  | No Data Mask Configuration.   |
|         | 20   | CFG_ENABLE_ECC_CODE_OVE_RWRITES | 0       | Read/Write  | Enable ECC Code Overwrite in Double Error Bit Detection.  |
| 0x043   | 7:0  | CFG_INTERFACE_WIDTH             | 0       | Read/Write  | Memory Interface Width.   |
| 0x044   | 3:0  | CFG_DEVICE_WIDTH                | 0       | Read/Write  | Memory Device Width.  |
| 0x045   | 0    | CFG_CAL_REQ                     | 0       | Read/Write  | Request re-calibration.   |
|         | 6:1  | CFG_CLOCK_OFF                   | 0       | Read/Write  | Disable memory clock.   |
| 0x047   | 0    | STS_CAL_SUCCESS                 | 0       | Read Only   | Calibration Success.  |
|         | 1    | STS_CAL_FAIL                    | 0       | Read Only   | Calibration Fail.   |
|         | 2    | STS_SBE_ERROR                   | 0       | Read Only   | Single Bit Error Detected.  |
|         | 3    | STS_DBE_ERROR                   | 0       | Read Only   | Double Bit Error Detected.  |
|         | 4    | STS_CORR_DROPPED                | 0       | Read Only   | Auto Correction Dropped.  |
| 0x048   | 0    | CFG_ENABLE_INTR                 | 0       | Read/Write  | Enable Interrupt  |
|         | 1    | CFG_MASK_SBE_INTR               | 0       | Read/Write  | Mask Single Bit Error Interrupt.  |
|         | 2    | CFG_MASK_DBE_INTR               | 0       | Read/Write  | Mask Double Bit Error Interrupt.  |
|         | 3    | CFG_MASK_DBE_INTR               | 0       | Write Clear | Clear Interrupt.  |
| 0x049   | 7:0  | STS_SBD_COUNT                   | 0       | Read Only   | Reports the number of SBE errors that have occurred since the status register counters were last cleared. |
| 0x04A   | 7:0  | STS_DBE_COUNT                   | 0       | Read Only   | Reports the number of SBE errors that have occurred since the status register counters were last cleared. |
| 0x04B   | 31:0 | STS_ERR_ADDR                    | 0       | Read Only   | The address of the most recent ECC error.   |
| 0x04F   | 0    | CFG_MASK_CORR_DROPPED_I_NTR     | 0       | Read/Write  | Auto Correction Dropped Count.  |
| 0x050   | 7:0  | CFG_MASK_CORR_DROPPED_I_NTR     | 0       | Read Only   | Auto Correction Dropped Count.  |
| 0x051   | 31:0 | STS_CORR_DROPPED_ADDR           | 0       | Read Only   | Auto Correction Dropped Address.  |
| 0x055   | 5:0  | CFG_STARVE_LIMIT                | 0       | Read/Write  | Starvation Limit.   |
| 0x056   | 1:0  | CFG_MEM_BL                      | 0       | Read/Write  | Burst Length.   |
|         | 2    | CFG_MEM_BL                      | 0       | Read/Write  | ECC Enable.   |
| 0x057   | 1:0  | CFG_MEM_BL                      | 0       | Read/Write  | Specifies controller interface width.   |
| 0x058   | 11:0 | CMD_PORT_WIDTH                  | 0       | Read/Write  | Specifies per command port data width.  |

*continued...*



| Address | Bit   | Name                  | Default | Access     | Description  |
|---------|-------|-----------------------|---------|------------|--|
| 0x05A   | 11:0  | CMD_FIFO_MAP          | 0       | Read/Write | Specifies command port to Write FIFO association.                    |
| 0x05C   | 11:0  | CFG_CPORT_RFIFO_MAP   | 0       | Read/Write | Specifies command port to Read FIFO association.                     |
|         | 23:12 | CFG_RFIFO_CPORT_MAP   | 0       | Read/Write | Port assignment (0 - 5) associated with each of the N FIFO.          |
|         | 31:24 | CFG_WFIFO_CPORT_MAP   | 0       | Read/Write | Port assignment (0 - 5) associated with each of the N FIFO. (con't)  |
| 0x05D   | 3:0   | CFG_WFIFO_CPORT_MAP   | 0       | Read/Write | Port assignment (0 - 5) associated with each of the N FIFO.          |
|         | 14:4  | CFG_CPORT_TYPE        | 0       | Read/Write | Command port type.   |
| 0x062   | 2:0   | CFG_CLOSE_TO_FULL     | 0       | Read/Write | Indicates when the FIFO has this many empty entries left.            |
|         | 5:3   | CFG_CLOSE_TO_EMPTY    | 0       | Read/Write | Indicates when the FIFO has this many valid entries left.            |
|         | 11:6  | CFG_CLOSE_TO_EMPTY    | 0       | Read/Write | Port works in synchronous mode.                                      |
|         | 12    | CFG_INC_SYNC          | 0       | Read/Write | Set the number of FF as clock synchronizer.                          |
| 0x067   | 5:0   | CFG_ENABLE_BONDING    | 0       | Read/Write | Enables bonding for each of the control ports.                       |
|         | 7:6   | CFG_DELAY_BONDING     | 0       | Read/Write | Set to the value used for the bonding input to bonding output delay. |
| 0x069   | 5:0   | CFG_AUTO_PCH_ENABLE   | 0       | Read/Write | Control auto-precharge options.                                      |
| 0x06A   | 17:0  | MP_SCHEDULER_PRIORITY | 0       | Read/Write | Set absolute user priority of the port                               |
| 0x06D   | 29:0  | RCFG_ST_WT            | 0       | Read/Write | Set static weight of the port.                                       |
|         | 31:30 | RCFG_SUM_PRI_WT       | 0       | Read/Write | Set the sum of static weights for particular user priority.          |
| 0x06E   | 31:0  | RCFG_SUM_PRI_WT       | 0       | Read/Write | Set the sum of static weights for particular user priority.          |
| 0x06F   | 29:0  | RCFG_SUM_PRI_WT       | 0       | Read/Write | Set the sum of static weights for particular user priority.          |
| 0x0B9   | 0     | CFG_DISABLE_MERGING   | 0       | Read/Write | Set to a one to disable command merging.                             |

## 8.6. Sequence of Operations

Various blocks pass information in specific ways in response to write, read, and read-modify-write commands.



### Write Command

When a requesting master issues a write command together with write data, the following events occur:

- The input interface accepts the write command and the write data.
- The input interface passes the write command to the command generator and the write data to the write data buffer.
- The command generator processes the command and sends it to the timing bank pool.
- Once all timing requirements are met and a write-data-ready notification has been received from the write data buffer, the timing bank pool sends the command to the arbiter.
- When rank timing requirements are met, the arbiter grants the command request from the timing bank pool and passes the write command to the AFI interface.
- The AFI interface receives the write command from the arbiter and requests the corresponding write data from the write data buffer.
- The PHY receives the write command and the write data, through the AFI interface.

### Read Command

When a requesting master issues a read command, the following events occur:

- The input interface accepts the read command.
- The input interface passes the read command to the command generator.
- The command generator processes the command and sends it to the timing bank pool.
- Once all timing requirements are met, the timing bank pool sends the command to the arbiter.
- When rank timing requirements are met, the arbiter grants the command request from the timing bank pool and passes the read command to the AFI interface.
- The AFI interface receives the read command from the arbiter and passes the command to the PHY.
- The PHY receives the read command through the AFI interface, and returns read data through the AFI interface.
- The AFI interface passes the read data from the PHY to the read data buffer.
- The read data buffer sends the read data to the master through the input interface.



### Read-Modify-Write Command

A read-modify-write command can occur when enabling ECC for partial write, and for ECC correction commands. When a read-modify-write command is issued, the following events occur:

- The command generator issues a read command to the timing bank pool.
- The timing bank pool and arbiter passes the read command to the PHY through the AFI interface.
- The PHY receives the read command, reads data from the memory device, and returns the read data through the AFI interface.
- The read data received from the PHY passes to the ECC block.
- The read data is processed by the write data buffer.
- When the write data buffer issues a read-modify-write data ready notification to the command generator, the command generator issues a write command to the timing bank pool. The arbiter can then issue the write request to the PHY through the AFI interface.
- When the PHY receives the write request, it passes the data to the memory device.

## 8.7. Document Revision History

| Date          | Version    | Changes  |
|---------------|------------|--|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release.   |
| May 2016      | 2016.05.02 | Maintenance release.   |
| November 2015 | 2015.11.02 | <ul style="list-style-type: none"><li>• Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i>.</li><li>• Added CFG_GEN_SBE and CFG_GEN_DBE to <i>Hard Controller Register Map</i> table.</li></ul>  |
| May 2015      | 2015.05.04 | Maintenance release.   |
| December 2014 | 2014.12.15 | <ul style="list-style-type: none"><li>• Renamed <i>Controller Register Map</i> to <i>Soft Controller Register Map</i>.</li><li>• Added <i>Hard Controller Register Map</i>.</li></ul>  |
| August 2014   | 2014.08.15 | <ul style="list-style-type: none"><li>• Added "asynchronous" to descriptions of mp_cmd_reset_n_#_reset_n, mp_rfifo_reset_n_#_reset_n, and mp_wfifo_reset_n_#_reset_n signals in the <i>MPFE Signals</i> table.</li><li>• Added <i>Reset</i> description to <i>Hard Memory Controller</i> section.</li><li>• Added full-rate hard memory controller information for Arria V and Cyclone V to description of avl_addr[ ] in the <i>Local Interface Signals</i> table.</li><li>• Reworded avl_burstbegin description in the <i>Local Interface Signals</i> table.</li></ul> |
| December 2013 | 2013.12.16 | <ul style="list-style-type: none"><li>• Removed references to ALTMEMPHY.</li><li>• Removed references to SOPC Builder.</li><li>• Removed Half-Rate Bridge information.</li><li>• Modified Burst Merging description.</li><li>• Expanded description of avl_ready in Local Interface Signals table.</li></ul>   |

*continued...*



| Date          | Version | Changes   |
|---------------|---------|---|
|               |         | <ul style="list-style-type: none"><li>Added descriptions of local_cal_success and local_cal_fail to Local Interface Signals table.</li><li>Modified description of avl_size in Local Interface Signals table.</li><li>Added guidance to initialize memory before use.</li></ul> |
| November 2012 | 2.1     | <ul style="list-style-type: none"><li>Added Controller Register Map information.</li><li>Added Burst Merging information.</li><li>Updated User-Controlled Refresh Interface information.</li><li>Changed chapter number from 4 to 5.</li></ul>                                  |
| June 2012     | 2.0     | <ul style="list-style-type: none"><li>Added LPDDR2 support.</li><li>Added Feedback icon.</li></ul>  |
| November 2011 | 1.1     | <ul style="list-style-type: none"><li>Revised Figure 5-1.</li><li>Added AXI to Avalon-ST Converter information.</li><li>Added AXI Data Slave Interface information.</li><li>Added Half-Rate Bridge information.</li></ul>   |

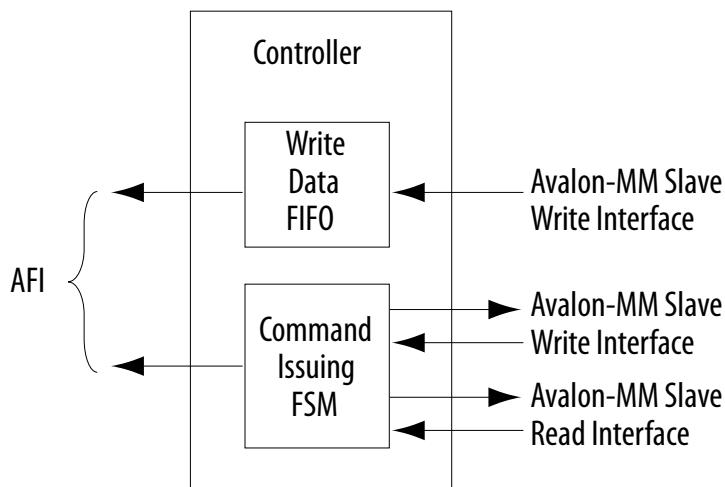
## 9. Functional Description—QDR II Controller

The QDR II, QDR II+, and QDR II+ Xtreme controller translates memory requests from the Avalon Memory-Mapped (Avalon-MM) interface to AFI, while satisfying timing requirements imposed by the memory configuration. QDR II, QDR II+, and QDR II+ Xtreme SRAM has unidirectional data buses, therefore read and write operations are highly independent of each other and each has its own interface and state machine.

### 9.1. Block Description

The following figure shows a block diagram of the QDR II, QDR II+, and QDR II+ Xtreme SRAM controller architecture.

**Figure 162. QDR II, QDR II+, and QDR II+ Xtreme SRAM Controller Architecture Block Diagram**



#### 9.1.1. Avalon-MM Slave Read and Write Interfaces

The read and write blocks accept read and write requests, respectively, from the Avalon-MM interface. Each block has a simple state machine that represents the state of the command and address registers, which stores the command and address when a request arrives.

The read data passes through without the controller registering it, as the PHY takes care of read latency. The write data goes through a pipeline stage to delay for a fixed number of cycles as specified by the write latency. In the full-rate burst length of four controller, the write data is also multiplexed into a burst of 2, which is then multiplexed again in the PHY to become a burst of 4 in DDR.



The user interface to the controller has separate read and write Avalon-MM interfaces because reads and writes are independent of each other in the memory device. The separate channels give efficient use of available bandwidth.

### 9.1.2. Command Issuing FSM

The command-issuing finite-state machine (FSM) has two states: INIT and INIT\_COMPLETE. In the INIT\_COMPLETE state, commands are issued immediately as requests arrive using combinational logic and do not require state transitions.

### 9.1.3. AFI

The QDR II, QDR II+, and QDR II+ Xtreme controller communicates with the PHY using the AFI interface.

In the full-rate burst-length-of-two configuration, the controller can issue both read and write commands in the same clock cycle. In the memory device, both commands are clocked on the positive edge, but the read address is clocked on the positive edge, while the write address is clocked on the negative edge. Care must be taken on how these signals are ordered in the AFI.

For the half-rate burst-length-of-four configuration, the controller also issues both read and write commands, but the AFI width is doubled to fill two memory clocks per controller clock. Because the controller issues only one write command and one read command per controller clock, the AFI read and write signals corresponding to the other memory cycle are tied to no operation (NOP).

For the full-rate burst-length-of-four configuration, the controller alternates between issuing read and write commands every clock cycle. The memory device requires two clock cycles to complete the burst-length-of-four operation and requires an interleaving of read and write commands.

For information on the AFI, refer to *AFI 4.0 Specification* in chapter 1, *Functional Description - UniPHY*.

#### Related Information

- [AFI 4.0 Specification](#) on page 117
- [Functional Description—UniPHY](#) on page 13

## 9.2. Avalon-MM and Memory Data Width

The following table lists the data width ratio between the memory interface and the Avalon-MM interface.

The half-rate controller does not support burst-of-2 devices because it under-uses the available memory bandwidth. Regardless of full or half-rate decision and the device burst length, the Avalon-MM interface must supply all the data for the entire memory burst in a single clock cycle. Therefore the Avalon-MM data width of the full-rate controller with burst-of-4 devices is four times the memory data width. For width-expanded configurations, the data width is further multiplied by the expansion factor (not shown in table 5-1 and 5-2).



**Table 101. Data Width Ratio**

| Memory Burst Length                               | Half-Rate Designs | Full-Rate Designs |
|---|-------------------|-------------------|
| QDR II 2-word burst                               | No Support        | 2:1               |
| QDR II, QDR II+, and QDR II+ Extreme 4-word burst |                   | 4:1               |

### 9.3. Signal Descriptions

The following tables lists the signals of the controller's Avalon-MM slave interface.

**Table 102. Avalon-MM Slave Read Signals**

| UniPHY Signal Name | Arria 10 Signal Name | Width (UniPHY)  | Width (Arria 10)  | Direction | Avalon-MM Signal Type |
|--------------------|----------------------|---|---|-----------|-----------------------|
| avl_r_ready        | amm_ready_0          | 1   | 1   | Out       | waitrequest_n         |
| avl_r_read_req     | amm_read_0           | 1   | 1   | In        | read                  |
| avl_r_addr         | amm_address_0        | 15–25   | 17–23   | In        | address               |
| avl_r_rdata_valid  | amm_readdatavalid_0  | 1   | 1   | Out       | readdatavalid         |
| avl_r_rdata        | amm_readdata_0       | 9, 18, 36 (or 8, 16, 32 if power-of-2-bus is enabled in the controller) | 9, 18, 36 (or 8, 16, 32 if power-of-2-bus is enabled in the controller) | Out       | readdata              |
| avl_r_size         | amm_burstcount_0     | $\log_2(\text{MAX_BURST_SIZE}) + 1$                                     | $\text{ceil}(\log_2(\text{CTRL_QDR2_AVL_MAX_BURST_COUNT} + 1))$         | In        | burstcount            |

**Note:** To obtain the actual signal width, you must multiply the widths in the above table by the data width ratio and the width expansion ratio.

**Table 103. Avalon-MM Slave Write Signals**

| UniPHY Signal Name | Arria 10 Signal Name | Width (UniPHY)  | Width (Arria 10)  | Direction | Avalon-MM Signal Type |
|--------------------|----------------------|---|---|-----------|-----------------------|
| avl_w_ready        | amm_ready_1          | 1   | 1   | Out       | waitrequest_n         |
| avl_w_write_req    | amm_write_1          | 1   | 1   | In        | write                 |
| avl_w_addr         | amm_address_1        | 15–25   | 17–23   | In        | address               |
| avl_w_wdata        | amm_writedata_1      | 9, 18, 36 (or 8, 16, 32 if power-of-2-bus is enabled in the controller) | 9, 18, 36 (or 8, 16, 32 if power-of-2-bus is enabled in the controller) | In        | writedata             |
| avl_w_be           | amm_bytewable_1      | 1,2,4   | 1,2,4   | In        | bytewable             |
| avl_w_size         | amm_burstcount_1     | $\log_2(\text{MAX_BURST_SIZE}) + 1$                                     | $\text{ceil}(\log_2(\text{CTRL_QDR2_AVL_MAX_BURST_COUNT} + 1))$         | In        | burstcount            |



**Note:** To obtain the actual signal width, you must multiply the widths in the above table by the data width ratio and the width expansion ratio.

#### Related Information

[Avalon Interface Specifications](#)

## 9.4. Document Revision History

| Date          | Version    | Changes  |
|---------------|------------|--|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release.   |
| May 2016      | 2016.05.02 | Maintenance release.   |
| November 2015 | 2015.11.02 | Maintenance release.   |
| May 2015      | 2015.05.04 | Maintenance release.   |
| December 2014 | 2014.12.15 | Maintenance release.   |
| August 2014   | 2014.08.15 | <ul style="list-style-type: none"><li>Added QDR II+ Xtreme throughout the chapter.</li><li>Added full-rate, burst-length-of-four information to <i>AFI</i> section.</li><li>Revised <i>Avalon-MM Slave Read Signals</i> table to include Arria 10 information.</li><li>Revised <i>Avalon-MM Slave Write Signals</i> table to include Arria 10 information.</li></ul> |
| December 2013 | 2013.12.16 | Removed references to SOPC Builder.  |
| November 2012 | 3.3        | Changed chapter number from 5 to 6.  |
| June 2012     | 3.2        | Added Feedback icon.   |
| November 2011 | 3.1        | Harvested Controller chapter from 11.0 <i>QDR II and QDR II+ SRAM Controller with UniPHY User Guide</i> .  |

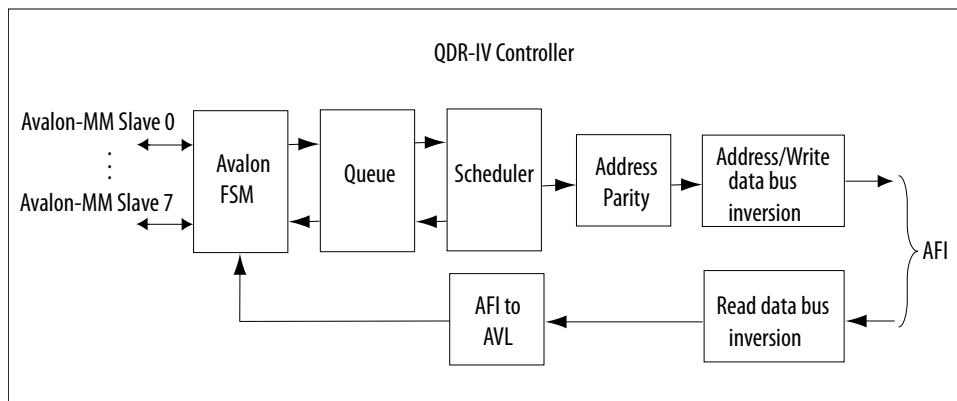
## 10. Functional Description—QDR-IV Controller

The QDR-IV controller translates memory requests from the Avalon Memory-Mapped (Avalon-MM) interface to AFI, while satisfying timing requirements imposed by the memory configuration. QDR-IV has two independent bidirectional data ports, each of which can perform read and write operations on each memory clock cycle, subject to bank address and bus turnaround restrictions. To maximize data bus utilization, the QDR-IV controller provides eight separate Avalon interfaces, one for each QDR-IV data port and time slot (at quarter rate).

### 10.1. Block Description

The following figure shows a block diagram of the QDR-IV controller architecture.

**Figure 163. QDR-IV Controller Architecture Block Diagram**



**Note:** To achieve maximum performance for a QDR-IV interface on an Arria 10 device, read data bus inversion must be enabled.

#### 10.1.1. Avalon-MM Slave Read and Write Interfaces

To maximize available bandwidth utilization, the QDR-IV external memory controller provides eight separate bidirectional Avalon interfaces—one channel for each of the two QDR-IV data ports in each of the four memory time slots. At the memory device, the PHY ensures that port A commands are issued on the rising edge of the clock, and port B commands are issued on the falling edge of the clock.

The Avalon finite state machine (FSM) implements the standard Avalon-MM interface for each channel. New commands arrive in a queue, before being sent to the PHY by the scheduler. The scheduler ensures that bank policy and bus turnaround restrictions are met. The controller handles address/data bus inversion, if you have enabled those features.

The controller schedules commands in a simple round-robin fashion, always maintaining the time slot relationship shown in the following table. At each AFI cycle, the scheduler attempts to issue up to eight commands—one from each channel. If a particular command cannot be issued due to bank policy or bus turnaround violations, only commands from the preceding channels are issued. The remaining commands are issued in the following one or more cycles, depending on whether further violations occur. Read data passes through without the controller registering it, after AFI to Avalon bus conversion. The PHY implements read latency requirements.

**Table 104. Avalon-MM to Memory Time Slot and Port Mapping**

| Avalon-MM Slave Interface | Memory Time Slot | Memory Port |
|---------------------------|------------------|-------------|
| 0                         | 0                | A           |
| 1                         | 0                | B           |
| 2                         | 1                | A           |
| 3                         | 1                | B           |
| 4                         | 2                | A           |
| 5                         | 2                | B           |
| 6                         | 3                | A           |
| 7                         | 3                | B           |

### 10.1.2. AFI

The QDR-IV controller communicates with the PHY using the AFI interface.

The controller supports a quarter-rate burst-length-of-two configuration, and can issue up to eight read and/or write commands per controller clock cycle. The controller may have to reduce the actual number of commands issued to avoid banking and bus turnaround violations. If you use QDR-IV devices with banked operation, you cannot access the same bank in the same memory clock cycle. Write latency is much shorter than read latency, therefore you must be careful not to place write data at the same time that read data is driven on the bus, on the same port. In addition, when switching between read and write operations, a delay may be required to avoid signal reflection and allow enough time for dynamic on-chip termination (OCT) to work. If necessary, the scheduler in the controller can delay the issuing of commands to avoid these issues.

For information on the AFI, refer to *AFI 4.0 Specification* in *Functional Description - Arria 10 EMIF IP*.

#### Related Information

- [AFI 4.0 Specification](#) on page 117
- [Functional Description—Intel Arria 10 EMIF IP](#) on page 141

## 10.2. Avalon-MM and Memory Data Width

The following table lists the data width ratio between the memory interface and the Avalon-MM interface.



QDR-IV memory devices use a burst length of 2. Because the Avalon-MM interface must supply all of the data for the entire memory burst in a single controller clock cycle, the Avalon-MM data width of the controller is double the memory data width. For width-expanded configurations, the data width is further multiplied by the expansion factor.

**Table 105. Data Width Ratio**

| Memory Burst Length | Controller |
|---------------------|------------|
| 2                   | 2:1        |

### 10.3. Signal Descriptions

The following tables lists the signals of the controller's Avalon-MM slave interface.

**Table 106. Avalon-MM Slave Interface Signals**

| Signal Name         | Width   | Direction | Avalon-MM Signal Type |
|---------------------|---|-----------|-----------------------|
| amm_ready_i         | 1   | Out       | waitrequest_n         |
| amm_read_i          | 1   | In        | read                  |
| amm_write_i         | 1   | In        | write                 |
| amm_address_i       | 21-25   | In        | address               |
| amm_readdatavalid_i | 1   | Out       | readdatavalid         |
| amm_readdata_i      | 18, 36, 72  | Out       | readdata              |
| amm_writedata_i     | 18, 36, 72  | In        | writedata             |
| amm_burstcount_i    | $\text{ceil}(\log_2(\text{CTRL_QDR4_AVL_MAX_BURST_COUNT}+1))$ | In        | burstcount            |
| emif_usr_clk        | 1   | Out       | clk                   |
| emif_usr_reset_n    | 1   | Out       | reset_n               |
| global_reset_n      | 1   | In        | reset_n               |

*Note:*

1. In the above table, the \_i suffix represents the Avalon-MM interface index, which has the range of 0-7.
2. To obtain the actual signal width for the data ports (readdata, writedata), you must multiply the widths in the above table by the data width ratio and the width expansion ratio.
3. • emif\_usr\_clk: User clock domain.  
 • emif\_usr\_reset\_n: Reset for the clock domain. Asynchronous assertion and synchronous deassertion (to the emif\_usr\_clk clock signal).  
 • global\_reset\_n: Asynchronous reset causes the memory interface to be reset and recalibrated.

**Related Information**[Avalon Interface Specifications](#)

## 10.4. Document Revision History

| Date          | Version    | Changes              |
|---------------|------------|----------------------|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release. |
| May 2016      | 2016.05.02 | Maintenance release. |
| November 2015 | 2015.11.02 | Maintenance release. |
| May 2015      | 2015.05.04 | Maintenance release. |
| December 2014 | 2014.12.15 | Initial Release.     |

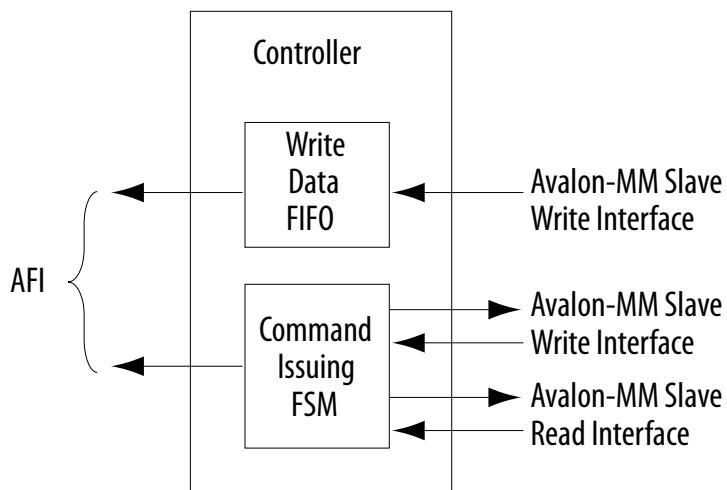
## 11. Functional Description—RLDRAM II Controller

The RLDARAM II controller translates memory requests from the Avalon Memory-Mapped (Avalon-MM) interface to AFI, while satisfying timing requirements imposed by the memory configurations.

### 11.1. Block Description

The following figure shows a block diagram of the RLDARAM II controller architecture.

**Figure 164. RLDARAM II Controller Architecture Block Diagram**



#### 11.1.1. Avalon-MM Slave Interface

The Avalon-MM slave interface accepts read and write requests. A simple state machine represents the state of the command and address registers, which stores the command and address when a request arrives.

The Avalon-MM slave interface decomposes the Avalon-MM address to the memory bank, column, and row addresses. The IP automatically maps the bank address to the LSB of the Avalon address vector.



The Avalon-MM slave interface includes a burst adaptor, which has two parts:

- The first part is a read and write request combiner that groups requests to sequential addresses into the native memory burst. Given that the second request arrives within the read and write latency window of the first request, the controller can combine and satisfy both requests with a single memory transaction.
- The second part is the burst divider in the front end of the Avalon-MM interface, which breaks long Avalon bursts into individual requests of sequential addresses, which then pass to the controller state machine.

### 11.1.2. Write Data FIFO Buffer

The write data FIFO buffer accepts write data from the Avalon-MM interface. The AFI controls the subsequent consumption of the FIFO buffer write data.

### 11.1.3. Command Issuing FSM

The command issuing finite-state machine (FSM) has three states.

The controller is in the `INIT` state when the PHY initializes the memory. Upon receiving the `afi_cal_success` signal, the state transitions to `INIT_COMPLETE`. If the calibration fails, `afi_cal_fail` is asserted and the state transitions to `INIT_FAIL`. The PHY receives commands only in the `INIT_COMPLETE` state.

When a refresh request arrives at the state machine at the same time as a read or write request, the refresh request takes precedence. The read or write request waits until there are no more refresh requests, and is issued immediately if timing requirements are met.

### 11.1.4. Refresh Timer

With automatic refresh, the refresh timer periodically issues refresh requests to the command issuing FSM. The refresh interval can be set at generation.

### 11.1.5. Timer Module

The timer module contains one DQ timer and eight bank timers (one per bank). The DQ timer tracks how often read and write requests can be issued, to avoid bus contention. The bank timers track the cycle time ( $t_{RC}$ ).

The 8-bit wide output bus of the bank timer indicates to the command issuing FSM whether each bank can be issued a read, write, or refresh command.

### 11.1.6. AFI

The RLDRAM II controller communicates with the PHY using the AFI interface. For information on the AFI, refer to *AFI 4.0 Specification* in *Functional Description - UniPHY*.

#### Related Information

- [AFI 4.0 Specification](#) on page 117
- [Functional Description—UniPHY](#) on page 13



## 11.2. User-Controlled Features

The **General Settings** tab of the parameter editor contains several features which are disabled by default. You can enable these features as required for your external memory interface.

### 11.2.1. Error Detection Parity

The error detection feature asserts an error signal if it detects any corrupted data during the read process.

The error detection parity protection feature creates a simple parity encoder block which processes all read and write data. For every 8 bits of write data, a parity bit is generated and concatenated to the data before it is written to the memory. During the subsequent read operation, the parity bit is checked against the data bits to ensure data integrity.

When you enable the error detection parity protection feature, the local data width is reduced by one. For example, a nine-bit memory interface will present eight bits of data to the controller interface.

You can enable error detection parity protection in the **Controller Settings** section of the **General Settings** tab of the parameter editor.

### 11.2.2. User-Controlled Refresh

The user-controlled refresh feature allows you to take control of the refresh process that the controller normally performs automatically. You can control when refresh requests occur, and, if there are multiple memory devices, you control which bank receives the refresh signal.

When you enable this feature, you disable auto-refresh, and assume responsibility for maintaining the necessary average periodic refresh rate. You can enable user-controlled refresh in the **Controller Settings** section of the **General Settings** tab of the parameter editor.

## 11.3. Avalon-MM and Memory Data Width

The following table lists the data width ratio between the memory interface and the Avalon-MM interface. The half-rate controller does not support burst-of-2 devices because it under-uses the available memory bandwidth.

**Table 107. Data Width Ratio**

| Memory Burst Length | Half-Rate Designs | Full-Rate Designs |
|---------------------|-------------------|-------------------|
| 2-word              | No Support        | 2:1               |
| 4-word              | 4:1               |                   |
| 8-word              |                   |                   |

## 11.4. Signal Descriptions

The following table lists the signals of the controller's Avalon-MM slave interface.



For information on the AFI signals, refer to *AFI 4.0 Specification* in *Functional Description - UniPHY*.

**Table 108. Avalon-MM Slave Signals**

| Signal          | Width           | Direction | Avalon-MM Signal Type | Description |
|-----------------|-----------------|-----------|-----------------------|-------------|
| avl_size        | 1 to 11         | In        | burstcount            | —           |
| avl_ready       | 1               | Out       | waitrequest_n         | —           |
| avl_read_req    | 1               | In        | read                  | —           |
| avl_write_req   | 1               | In        | write                 | —           |
| avl_addr        | □ 25            | In        | address               | —           |
| avl_rdata_valid | 1               | Out       | readdatavalid         | —           |
| avl_rdata       | 18, 36, 72, 144 | Out       | readdata              | —           |
| avl_wdata       | 18, 36, 72, 144 | In        | writedata             | —           |

**Note:** If you are using Qsys, the data width of the Avalon-MM interface is restricted to powers of two. Non-power-of-two data widths are available with the IP Catalog.

**Note:** The RLDRAM II controller does not support the byteenable signal. If the RLDRAM II controller is used with the Avalon-MM Efficiency Monitor and Protocol Checker, data corruption can occur if the byteenable signal on the efficiency monitor is used. For example, this can occur if using the JTAG Avalon-MM Master component to drive the efficiency monitor.

#### Related Information

- [AFI 4.0 Specification](#) on page 117
- [Functional Description—UniPHY](#) on page 13

### 11.5. Document Revision History

| Date          | Version    | Changes   |
|---------------|------------|---|
| May 2017      | 2017.05.08 | Rebranded as Intel.                               |
| October 2016  | 2016.10.31 | Maintenance release.                              |
| May 2016      | 2016.05.02 | Maintenance release.                              |
| November 2015 | 2015.11.02 | Maintenance release.                              |
| May 2015      | 2015.05.04 | Maintenance release.                              |
| December 2014 | 2014.12.15 | Maintenance release.                              |
| August 2014   | 2014.08.15 | Removed occurrence of MegaWizard Plug-In Manager. |
| December 2013 | 2013.12.16 | Removed references to SOPC Builder.               |

*continued...*



| Date          | Version | Changes  |
|---------------|---------|--|
| November 2012 | 3.3     | Changed chapter number from 6 to 7.  |
| June 2012     | 3.2     | Added Feedback icon.   |
| November 2011 | 3.1     | Harvested Controller chapter from 11.0 <i>RLDRAM II Controller with UniPHY IP User Guide</i> . |

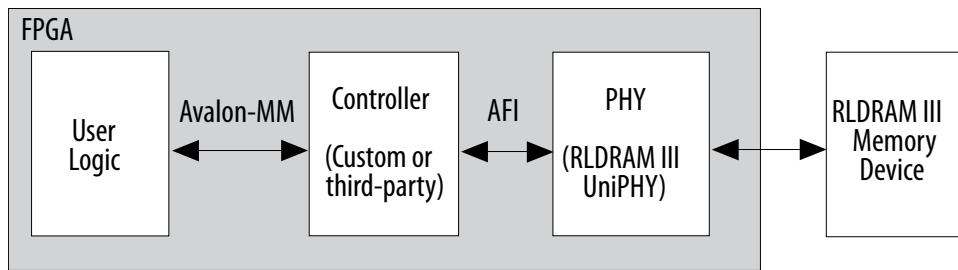
## 12. Functional Description—RLDRAM 3 PHY-Only IP

The RLDRAM 3 PHY-only IP works with a customer-supplied memory controller to translate memory requests from user logic to RLDRAM 3 memory devices, while satisfying timing requirements imposed by the memory configurations.

### 12.1. Block Description

The RLDRAM 3 UniPHY-based IP is a PHY-only offering which you can use with a third-party controller or a controller that you develop yourself. The following figure shows a block diagram of the RLDRAM 3 system architecture.

**Figure 165. RLDRAM 3 System Architecture**



### 12.2. Features

The RLDRAM 3 UniPHY-based IP supports features available from major RLDRAM 3 device vendors at speeds of up to 800 MHz.

The following list summarizes key features of the RLDRAM 3 UniPHY-based IP:

- support for Arria V GZ and Stratix V devices
- standard AFI interface between the PHY and the memory controller
- quarter-rate and half-rate AFI interface
- maximum frequency of 533 MHz for half-rate operation and 800 MHz for quarter-rate operation
- burst length of 2, 4, or 8
- x18 and x36 memory organization
- common I/O device support
- nonmultiplexed addressing
- multibank write and refresh protocol (programmable through mode register)
- optional use of data mask pins

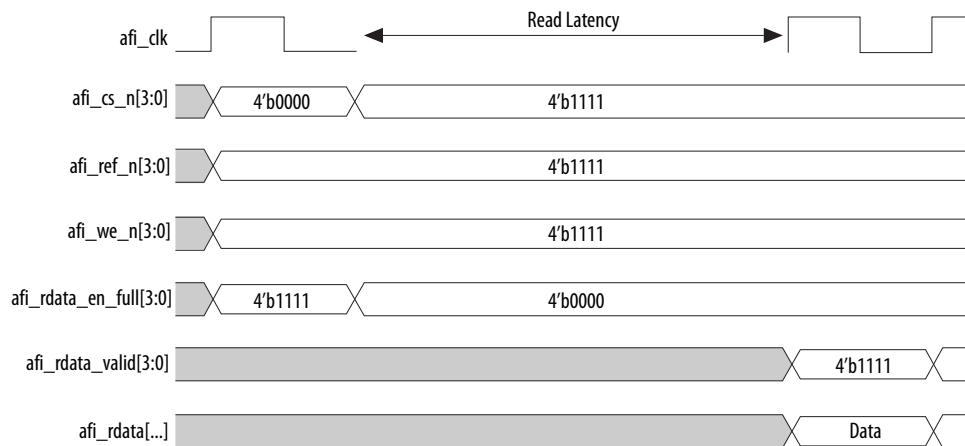


## 12.3. RLDRAM 3 AFI Protocol

The RLDRAM 3 UniPHY-based IP communicates with the memory controller using an AFI interface that follows the AFI 4.0 specification. To maximize bus utilization efficiency, the RLDRAM 3 UniPHY-based IP can issue multiple memory read/write operations within a single AFI cycle.

The following figure illustrates AFI bus activity when a quarter-rate controller issues four consecutive burst-length 2 read requests.

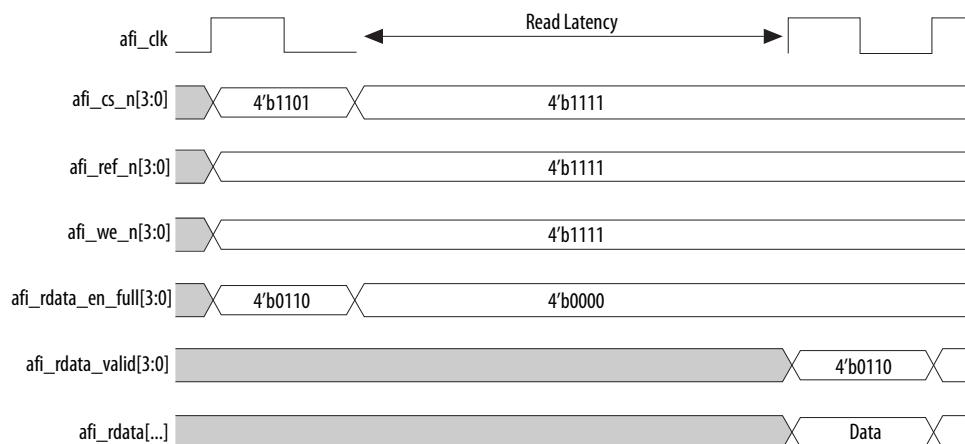
**Figure 166. AFI Bus Activity for Quarter-Rate Controller Issuing Four Burst-Length 2 Read Requests**



The controller does not have to begin a read or write command using channel 0 of the AFI bus. The flexibility afforded by being able to begin a command on any bus channel can facilitate command scheduling in the memory controller.

The following figure illustrates the AFI bus activity when a quarter-rate controller issues a single burst-length 4 read command to the memory on channel 1 of the AFI bus.

**Figure 167. AFI Bus Activity for Quarter-Rate Controller Issuing One Burst-Length 4 Read Request**





**Note:** For information on the AFI, refer to *AFI 4.0 Specification* in *Functional Description - UniPHY*.

#### Related Information

- [AFI 4.0 Specification](#) on page 117
- [Functional Description—UniPHY](#) on page 13

### 12.4. RLDRAM 3 Controller with Arria 10 EMIF Interfaces

The following table lists the RLDRAM 3 signals available for each interface when using Arria 10 EMIF IP.

| Signal                        | Interface Type | Description  |
|-------------------------------|----------------|--|
| <b>pll_ref_clk interface</b>  |                |  |
| pll_ref_clock                 | Clock input    | Clock input to the PLL inside EMIF.                              |
| <b>afi_clk_interface</b>      |                |  |
| afi_clock                     | Clock output   | AFI clock output from the PLL inside EMIF.                       |
| <b>afi_half_clk_interface</b> |                |  |
| afi_half_clock                | Clock output   | AFI clock output from the PLL inside EMIF running at half speed. |
| <b>Memory interface</b>       |                |  |
| mem_a                         | Conduit        | Interface signal between the PHY and the memory device.          |
| mem_ba                        |                |  |
| mem_ck                        |                |  |
| mem_ck_n                      |                |  |
| mem_cs_n                      |                |  |
| mem_dk                        |                |  |
| mem_dk_n                      |                |  |
| mem_dm                        |                |  |
| mem_dq                        |                |  |
| mem_qk                        |                |  |
| mem_qk_n                      |                |  |
| mem_ref_n                     |                |  |
| mem_we_n                      |                |  |
| mem_reset_n                   |                |  |
| <b>Status interface</b>       |                |  |
| local_init_done               | Conduit        | Memory interface status signal.                                  |
| local_cal_success             |                |  |
| local_cal_fail                |                |  |
| local_cal_request             |                |  |

*continued...*



| Signal               | Interface Type  | Description  |
|----------------------|-----------------|--|
| <b>oct_interface</b> |                 |  |
| oct_rzqin            | Conduit         | OCT reference resistor pins for RZQ.   |
| <b>afi_interface</b> | Avalon-MM slave | Altera PHY interface (AFI) signal between the PHY and the memory controller. |
| afi_addr             |                 |  |
| afi_ba               |                 |  |
| afi_cs_n             |                 |  |
| afi_we_n             |                 |  |
| afi_ref_n            |                 |  |
| afi_wdata_valid      |                 |  |
| afi_wdata            |                 |  |
| afi_dm               |                 |  |
| afi_rdata            |                 |  |
| afi_rdata_en_full    |                 |  |
| afi_rdata_valid      |                 |  |
| afi_RST_n            |                 |  |
| afi_cal_success      |                 |  |
| afi_cal_fail         |                 |  |
| afi_wlat             |                 |  |
| afi_rlat             |                 |  |

## 12.5. Document Revision History

| Date          | Version    | Changes  |
|---------------|------------|--|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release.   |
| May 2016      | 2016.05.02 | Maintenance release.   |
| November 2015 | 2015.11.02 | Maintenance release.   |
| May 2015      | 2015.05.04 | Maintenance release.   |
| December 2014 | 2014.12.15 | Maintenance release.   |
| August 2014   | 2014.08.15 | Added <i>RLDRAM 3 Controller with Arria 10 EMIF Interfaces</i> . |
| December 2013 | 2013.12.16 | Maintenance release.   |
| November 2012 | 1.0        | Initial release.   |

## 13. Functional Description—Example Designs

Generation of your external memory interface IP creates two independent example designs. These example designs illustrate how to instantiate and connect the memory interface for both synthesis and simulation flows.

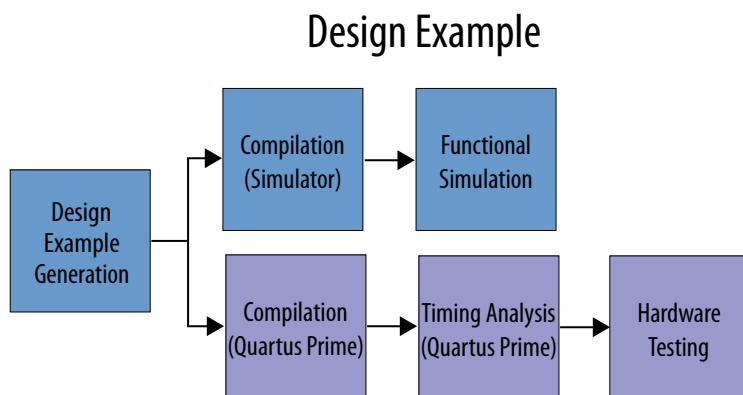
### 13.1. Arria 10 EMIF IP Example Designs Quick Start Guide

A new interface and more automated design example flow is available for Arria 10 external memory interfaces.

The **Example Designs** tab is available in the parameter editor when you specify an Arria 10 target device. This tab allows you to select from a list of presets for Intel FPGA development kits and target interface protocols. All tabs are automatically parameterized with appropriate values, based on the preset that you select. You can specify that the system create directories for simulation and synthesis file sets, and generate the file sets automatically.

You can generate an example design specifically for an Intel FPGA development kit, or for any EMIF IP that you generate.

**Figure 168. Using the Example Design**



#### 13.1.1. Typical Example Design Workflow

When you generate your EMIF IP, the system creates an example design that includes a traffic generator and external memory interface controller.



Figure 169. Example Design for Synthesis

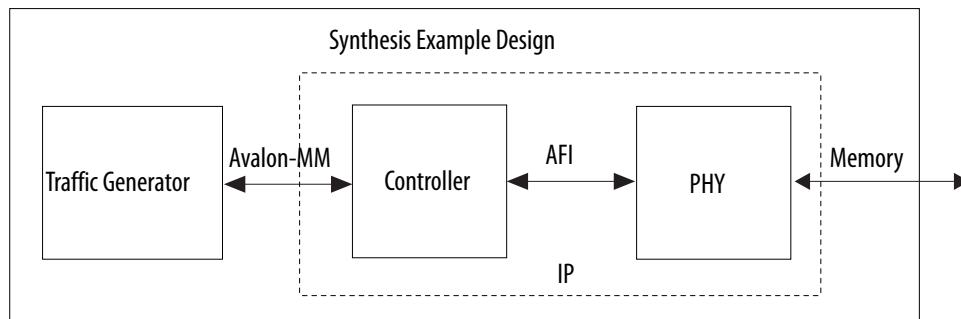
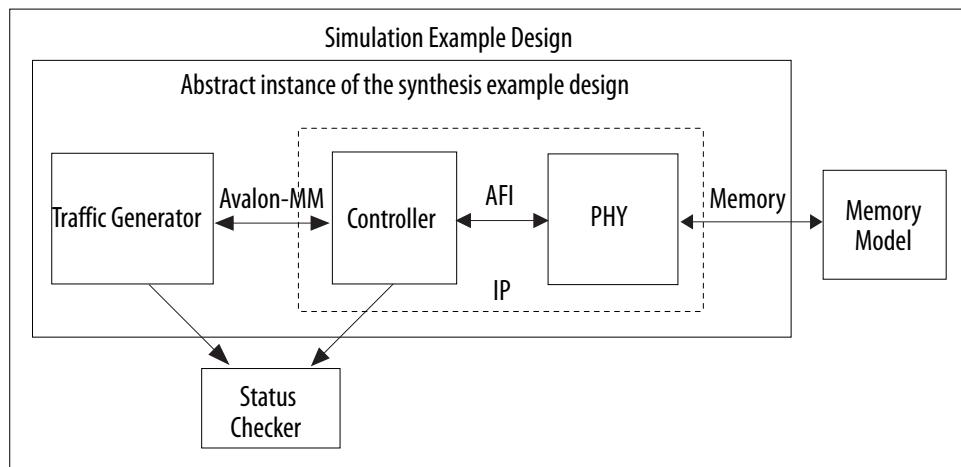
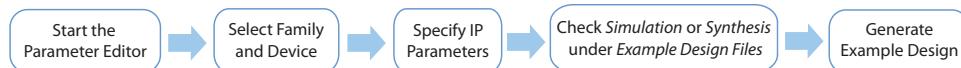


Figure 170. Example Design for Simulation



### Workflow Overview

Figure 171. Generating a Simulation or Synthesis File set

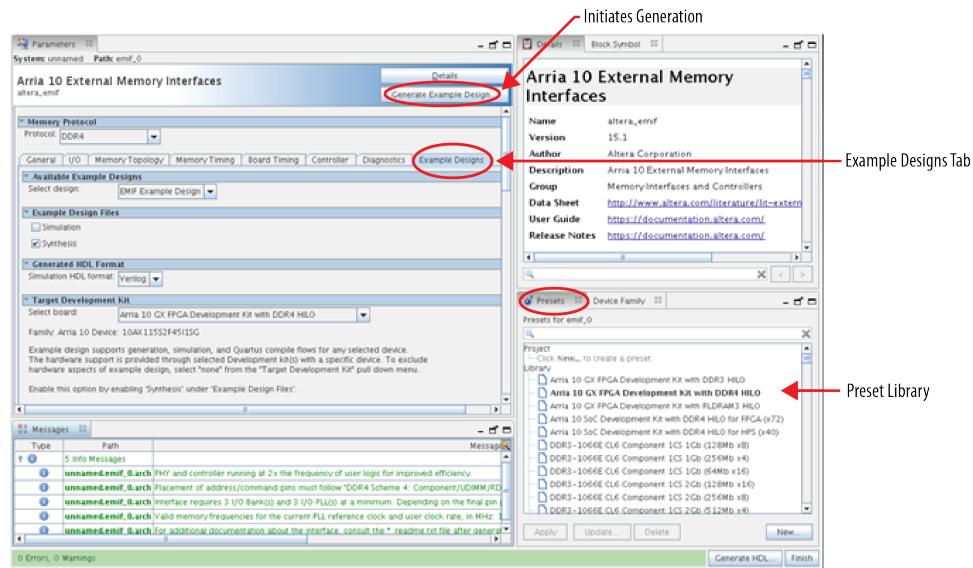


If you select **Simulation** or **Synthesis** under **Example Design Files** on the **Example Designs** tab, the system creates a complete simulation file set or a complete synthesis file set, in accordance with your selection.

### 13.1.2. Example Designs Interface Tab

If you have selected an Arria 10 device, the parameter editor includes an **Example Designs** tab which allows you to parameterize and generate your example designs.

**Figure 172. Example Designs Tab in the External Memory Interfaces Parameter Editor**



### Available Example Designs Section

The **Select design** pulldown allows you to select the desired example design. At present, **EMIF Example Design** is the only available choice, and is selected by default.

### Example Design Files Section

The example design supports generation, simulation, and Quartus Prime compilation flows for any selected device. The **Simulation** and **Synthesis** checkboxes in this section allow you to specify whether to generate a simulation file set, a synthesis file set, or both. Check **Simulation** to use the example design for simulation, or check **Synthesis** to use the example design for compilation and hardware. The system creates the specified file sets when you click the **Generate Example Design** button.

**Note:**

If you don't select the **Simulation** or **Synthesis** checkbox, the destination directory will contain Qsys design files, which are not compilable by Quartus Prime directly, but can be viewed or edited under Qsys.

- To create a compilable project, you must run the `quartus_sh -t make_qii_design.tcl` script in the destination directory.
- To create a simulation project, you must run the `quartus_sh -t make_sim_design.tcl` script in the destination directory.

### Generated HDL Format Section

The **Simulation HDL format** pulldown allows you to specify the target format for the generated design. At present, only Verilog filesets are supported.



### Target Development Kit Section

The **Select board** pulldown in this section applies the appropriate development kit pin assignments to the example design.

- This setting is available only when you turn on the **Synthesis** checkbox in the **Example Design Files** section.
- This setting must match the applied development kit present, or else an error message appears.

If the value *None* appears in the **Select board** pulldown, it indicates that the current parameter selections do not match any development kit configurations. You may apply a development kit-specific IP and related parameter settings by selecting one of the presets from the preset library. When you apply a preset, the current IP and other parameter settings are set to match the selected preset. If you want to save your current settings, you should do so before you select a preset. If you do select a preset without saving your prior settings, you can always save the new preset settings under a different name.

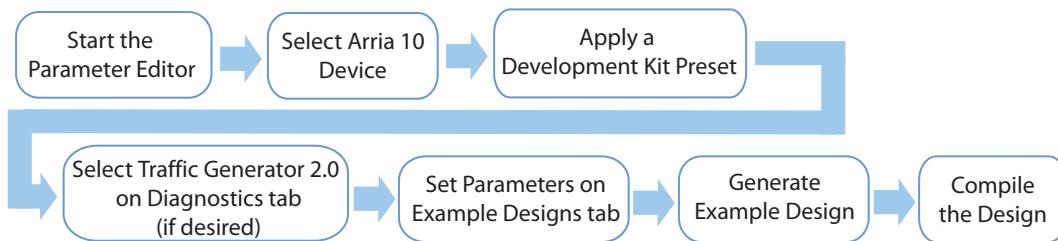
If you want to generate the example design for use on your own board, set **Select board** to *None*, generate the example design, and then add pin location constraints.

**Note:** In the 15.1 release, programming file (.sof/.pof) generation is supported for Arria 10 engineering samples only. If you would like to learn more about support for Arria 10 production devices in 15.1, contact your Intel FPGA representative or use the support link on [www.altera.com](http://www.altera.com).

### 13.1.3. Development Kit Preset Workflow

Arria 10 users can take advantage of development kit presets to create an example design that is automatically configured and ready for hardware testing or simulation. The use of presets greatly simplifies the workflow and reduces the steps necessary to have an EMIF example design working on an Arria 10 development kit.

**Figure 173. Generating an Example Design for Use With an Arria 10 Development Kit**



Follow these steps to generate the example design:

1. Under **Memory Interfaces and Controllers** (Tools > Qsys>IP Catalog>Library>Memory Interfaces and Controllers), select **Arria 10 External Memory Interfaces**. The parameter editor appears.
2. Specify a top-level name and the folder for your custom IP variation, and specify an Arria 10 device. Click **OK**.
3. Select the appropriate preset for your development kit from the **Presets Library**. Click **Apply**.

4. By default, the example design will be generated with the traditional Traffic Generator. If you would rather use the new EMIF Configurable Traffic Generator 2.0, select **Use configurable Avalon traffic generator 2.0** on the **Diagnostics** tab of the parameter editor.
5. On the **Example Designs** tab in the parameter editor, make the following settings:
  - a. Under **Available Example Designs**, only **EMIF Example Design** is available, and is selected by default.
  - b. Under **Example Design Files**, select **Simulation** or **Synthesis** to have the system create a simulation file set or synthesis file set.
  - c. Under **Generated HDL Format**, only **Verilog** is available, and is selected by default.
  - d. Under **Target Development Kit**, select the desired Arria 10 development kit. The development kit that you select here must match the development kit preset that you selected in step 1, or else an error message appears.

*Note:* The **Target Development Kit** section is available only if you have specified **Synthesis** under **Example Design Files**.
6. Click the **Generate Example Design** button at the top of the parameter editor. The software generates all files necessary to run simulations and hardware tests on the selected development kit.

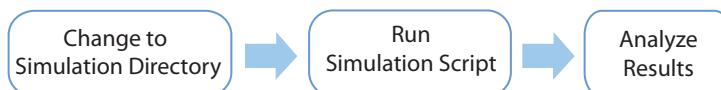
#### Related Information

Support Center Web Page on [www.altera.com](http://www.altera.com)

### 13.1.4. Compiling and Simulating the Design

This section explains how to simulate your EMIF example design.

**Figure 174. Procedure for Simulating the Design**



1. Change to the simulation directory.
2. Run the simulation script for the simulator of your choice. Refer to the table below.
3. Analyze the results.

**Table 109. Steps to Run Simulation**

| Simulator   | Working Directory                          | Instructions   |
|-------------|--|--|
| Riviera-PRO | <variation_name>_example_design\siml\aldec | To simulate the example design using the Riviera-PRO simulator, follow these steps:<br><br><i>continued...</i> |



| Simulator | Working Directory                                   | Instructions  |
|-----------|---|---|
|           |   | <ul style="list-style-type: none"> <li>a. At the Linux* or Windows* shell command prompt, change directory to: &lt;variation_name&gt;_example_design\sim\aldec.</li> <li>b. Execute the rivierapro_setup.tcl script by typing the following command at the Linux or Windows command prompt: vsim -do rivierapro_setup.tcl .</li> <li>c. To compile and elaborate the design after the script loads, type: ld_debug.</li> <li>d. Type run -all to run the simulation.</li> <li>e. A successful simulation ends with the following message, "--- SIMULATION PASSED ---"</li> </ul>  |
| NCSim     | <variation_name>_example_design\sim\cadence\        | <p>To simulate the example design using the NCSim simulator, follow these steps:</p> <ul style="list-style-type: none"> <li>a. At the Linux shell command prompt, change directory to &lt;variation_name&gt;_example_design\sim\cadence\.</li> <li>b. Run the simulation by typing the following at the command prompt: sh ncsim_setup.sh.</li> <li>c. A successful simulation ends with the following message, "--- SIMULATION PASSED ---"</li> </ul>  |
| ModelSim* | <variation_name>_example_design\sim\mentor\         | <p>To simulate the example design using the Modelsim simulator, follow these steps:</p> <ul style="list-style-type: none"> <li>a. At the Linux or Windows shell command prompt, change directory to &lt;variation_name&gt;_example_design\sim\mentor\.</li> <li>b. Perform one of the following: <ul style="list-style-type: none"> <li>• Execute the msim_setup.tcl script by typing the following command at the Linux or Windows command prompt: vsim -do msim_setup.tcl.</li> <li>• Type the following command at the ModelSim command prompt: do msim_setup.tcl.</li> </ul> </li> <li>c. A successful simulation ends with the following message, "--- SIMULATION PASSED ---"</li> </ul> |
| VCS       | <variation_name>_example_design\sim\synopsys\vcsmx\ | <p>To simulate the example design using the VCS simulator, follow these steps:</p> <ul style="list-style-type: none"> <li>a. At the Linux shell command prompt, change directory to &lt;variation_name&gt;_example_design\sim\synopsys\vcsmx\.</li> <li>b. Run the simulation by typing the following at the command prompt: sh vcsmx_setup.sh.</li> <li>c. A successful simulation ends with the following message, "--- SIMULATION PASSED ---"</li> </ul>   |

### 13.1.5. Compiling and Testing the Design in Hardware

This section explains how to compile and test your EMIF example design in hardware.

**Figure 175. Procedure for Testing in Hardware**



Follow these steps to compile and test the design in hardware:

1. Before compiling a design, select a method for controlling and monitoring the following pins:



- global\_reset\_reset\_n
  - <variation\_name>\_status\_local\_cal\_fail
  - <variation\_name>\_status\_local\_cal\_success
  - <variation\_name>\_tg\_0\_traffic\_gen\_fail
  - <variation\_name>\_tg\_0\_traffic\_gen\_pass
  - <variation\_name>\_tg\_0\_traffic\_gen\_timeout
  - Additional pins that may exist if the example design contains more than one traffic generator.
2. Assign pins by one of the following means, as appropriate for your situation:
- If you have selected a development kit preset, the above pins are automatically assigned to switches and LEDs on the board. See the documentation accompanying your specific development kit for details.
  - If you are using a non-development kit board, assign the above pins to appropriate locations on your board.
  - Alternatively, the above pins can be controlled and monitored using Intel in-system sources and probes. To instantiate these in your design, add the following line to your (.qsf) file: set\_global\_assignment -name VERILOG\_MACRO "ALTERA\_EMIF\_ENABLE\_ISSP=1"
3. Compile the design by clicking **Processing > Start Compilation** in the Quartus Prime software.
4. Configure the FPGA on the board using the generated (.sof) file.
5. Issue a reset on the global\_reset\_reset\_n port. If the traffic generator test completes successfully, the <variation\_name>\_status\_local\_cal\_success and <variation\_name>\_tg\_0\_traffic\_gen\_pass signals will go high, indicating success.
6. If you are using the Traffic Generator 2.0, you can configure and monitor the traffic generator using the EMIF Debug Toolkit. Refer to *Configuring the Traffic Generator 2.0*.

## 13.2. Testing the EMIF Interface Using the Traffic Generator 2.0

The EMIF Configurable Traffic Generator 2.0 can assist in debugging and stress-testing your external memory interface. The traffic generator 2.0 supports Arria 10 and later device families.

### Key Features

The traffic generator 2.0 has the following key features:

- Is a standalone, soft-logic device that resides in the FPGA core.
- Is independent of the FPGA architecture and the external memory protocol in use.
- Offers configuration options for the generation of reads and writes, addressing, data, and data mask.

For information on using the Traffic Generator 2.0 from within the EMIF Debug Toolkit, refer to *Traffic Generator 2.0 in External Memory Interface Debug Toolkit*.



### 13.2.1. Configurable Traffic Generator 2.0 Configuration Options

The EMIF Configurable Traffic Generator 2.0 offers a range of configuration options for fast debugging of your external memory interface. You configure the traffic generator by modifying address-mapped registers through the simulation test bench file or by creating a configuration test stage.

#### Configuration Syntax

The test bench includes an example test procedure.

The syntax for writing to a configuration register is as follows:

```
tg_send_cfg_write_<index>(<Register Name>, <Value to be written>);
```

The *index* value represents the index of the memory interface (which is usually 0, but can be 0/1 in ping-pong PHY mode).

The *register name* values are listed in the tables of configuration options, and can also be found in the `altera_emif_avl_tg_defs.sv` file, in the same directory as the test bench.

The final configuration command must be a write of any value to `TG_START`, which starts the traffic generator for the specified interface.

#### Configuration Options

Configuration options are divided into read/write, address, data, and data mask generation categories.

**Table 110. Configuration Options for Read/Write Generation**

| Toolkit GUI Tab | Toolkit GUI Parameter | Register Name         | Description  |
|-----------------|-----------------------|-----------------------|--|
| --              | --                    | TG_START              | Tells the system to perform a write to this register to initiate traffic generation test.  |
| Loops           | Loops                 | TG_LOOP_COUNT         | Specifies the number of read/write loops to perform before completion of the test stage. A loop is a single iteration of writes and reads. Upon completion, the base address is either incremented (SEQ or RAND_SEQ) or replaced by a newly generated random address (RAND or RAND_SEQ). |
|                 | Writes per block      | TG_WRITE_COUNT        | Specifies the number of writes to be performed in each loop of the test.   |
|                 | Reads per block       | TG_READ_COUNT         | Specifies the number of reads to be performed in each loop of the test. This register must have the same value as <code>TG_WRITE_COUNT</code> .  |
|                 | --                    | TG_WRITE_REPEAT_COUNT | Specifies the number of times each write transaction is repeated.  |
|                 | --                    | TG_READ_REPEAT_COUNT  | Specifies the number of times each read transaction is repeated.   |

*continued...*



| Toolkit GUI Tab | Toolkit GUI Parameter | Register Name       | Description  |
|-----------------|-----------------------|---------------------|--|
|                 | Reads per block       | TG_BURST_LENGTH     | Configures the length of each write/read burst to memory.  |
| --              | --                    | TG_CLEAR_FIRST_FAIL | Clears the record of first failure occurrence. New failure information such as expected data, read data, and fail address, is written to the corresponding registers following the next failure.     |
| --              | --                    | TG_TEST_BYTEEN      | Toggles the byte-enable (data mask enable) register within the traffic generator which allows the current test to use byte-enable signals.   |
| --              | --                    | TG_DATA_MODE        | Specifies the source of data used for data signal generation during the test. Set to 0 to use pseudo-random data. Set to 1 to use user-specified values stored in the static data generators.        |
| --              | --                    | TG_BYTEEN_MODE      | Specifies the source of data used for byte-enable signal generation during the test. Set to 0 to use pseudo-random data. Set to 1 to use user-specified values stored in the static data generators. |

**Table 111. Configuration Options for Address Generation**

| Toolkit GUI Tab | Toolkit GUI Parameter        | Register Name           | Description   |
|-----------------|------------------------------|-------------------------|---|
| Address         | Start Address                | TG_SEQ_START_ADDR_WR_L  | Specifies the sequential start address (lower 32 bits).   |
|                 | Start address                | TG_SEQ_START_ADDR_WR_H  | Specifies the sequential start address (upper 32 bits)  |
|                 | Address mode                 | TG_ADDR_MODE_WR         | Specifies how write addresses are generated by writing the value in parentheses to the register address. Values are: randomize the address for every write (0), increment sequentially from a specified address (1), increment sequentially from a random address (2), or perform one hot addressing (3). |
|                 | Number of sequential address | TG_RAND_SEQ_ADDRS_WR    | Specifies the number of times to increment sequentially on the random base address before generating a new random write address.  |
|                 | Return to start address      | TG_RETURN_TO_START_ADDR | Return to start address in deterministic sequential address mode (if 1 is written to TG_ADDR_MODE_WR) after every loop of transactions.   |
|                 | Rank   Mask Mode             | TG_RANK_MASK_EN         | Specifies the rank masking mode by writing the value in parentheses to the register address. Values are: disable rank masking (0),  |

***continued...***



| Toolkit GUI Tab | Toolkit GUI Parameter        | Register Name          | Description  |
|-----------------|------------------------------|------------------------|--|
|                 |                              |                        | maintain a static rank mask (1), cycle through rank masks incrementally (2).   |
|                 | Bank Address   Mask Mode     | TG_BANK_MASK_EN        | Specifies the bank masking mode by writing the value in parentheses to the register address. Values are: disable bank masking (0), maintain a static bank mask (1), cycle through bank masks incrementally (2), cycle through only three consecutive bank masks (3). |
|                 | Row   Mask Mode              | TG_ROW_MASK_EN         | Specifies the mode for row masking by writing the value in parentheses to the register address. Values are: disable row masking (0), maintain a static row mask (1), cycle through row masks incrementally (2).  |
|                 | Bank Group   Mask Mode       | TG_BG_MASK_EN          | Specifies the mode for bank group masking by writing the value in parentheses to the register address. Values are: disable bank group masking (0), maintain a static bank group mask (1), cycle through bank group masks incrementally (2).                          |
|                 | Rank   Mask Value            | TG_RANK_MASK           | Specifies the initial rank to be masked into the generated traffic generator address.  |
|                 | Bank Address   Mask Value    | TG_BANK_MASK           | Specifies the initial bank to be masked into the generated traffic generator address.  |
|                 | Row   Mask Value             | TG_ROW_MASK            | Specifies the initial row to be masked into the generated traffic generator address.   |
|                 | Bank Group   Mask Value      | TG_BG_MASK             | Specifies the initial bank group to be masked into the generated traffic generator address.  |
|                 | Sequential Address Increment | TG_SEQ_ADDR_INCR       | Specifies the increment to use when sequentially incrementing the address. This value is used by both deterministic sequential addressing and random sequential addressing. (Refer to TG_ADDR_MODE_WR)   |
|                 | Start Address                | TG_SEQ_START_ADDR_RD_L | Specifies the sequential start read address (lower 32 bits).   |
|                 |                              | TG_SEQ_START_ADDR_RD_H | Specifies the sequential start read address (upper 32 bits).   |
|                 | Address Mode                 | TG_ADDR_MODE_RD        | Similar to TG_ADDR_MODE_WR but for reads.  |
|                 | Number of sequential address | TG_RAND_SEQ_ADDRS_RD   | Specifies the number of times to increment the random sequential read address.   |

**Table 112. Configuration Options for Data Generation**

| Toolkit GUI Tab | Toolkit GUI Parameter                              | Register Name | Description   |
|-----------------|--|---------------|---|
| Data            | Seed/Fixed Pattern                                 | TG_DATA_SEED  | Specifies an initial value to the data generator corresponding to the index value.  |
|                 | <b>PRBS</b> and <b>Fixed Pattern</b> radio buttons | TG_DATA_MODE  | Specifies whether to treat the initial value of the data generator corresponding index as a seed for generating pseudo-random data (value of 0) or to keep the initial value static (value of 1). |

**Table 113. Configuration Options for Data Mask Generation**

| Toolkit GUI Tab | Toolkit GUI Parameter                              | Register Name  | Description   |
|-----------------|--|----------------|---|
| Data            | Seed/Fixed Pattern                                 | TG_BYTEEN_SEED | Specifies an initial value to the byte-enable generator corresponding to the index value.   |
|                 | <b>PRBS</b> and <b>Fixed Pattern</b> radio buttons | TG_BYTEEN_MODE | Specifies whether to treat the initial value of the byte-enable generator of corresponding index as a seed and generate pseudo-random data (value of 0) or to keep the initial value static (value of 1). |

### 13.2.1.1. Test Information

In the test bench file, register reads are encoded in a similar syntax to register writes.

The following example illustrates the syntax of a register read:

```
integer <Variable Name>;
tg_send_cfg_read_<index>(<Register Name>, <Variable Name>);
```

In hardware, you can probe the registers storing the test information (such as pnf per bit persist, first fail read address, first fail read data, and first fail expected data).

**Table 114. Test Information Read-Accessible Through Register Addresses**

| Register Name          | Description   |
|------------------------|---|
| TG_PASS                | Returns a high value if the traffic generator passes at the end of all the test stages. |
| TG_FAIL                | Returns a high value if the traffic generator fails at the end of all the test stages.  |
| TG_FAIL_COUNT_L        | Reports the failure count (lower 32 bits).  |
| TG_FAIL_COUNT_H        | Reports the failure count (upper 32 bits).  |
| TG_FIRST_FAIL_ADDR_L   | Reports the address of the first failure (lower 32 bits).                               |
| TG_FIRST_FAIL_ADDR_H   | Reports the address of the first failure (upper 32 bits).                               |
| TG_FIRST_FAIL_IS_READ  | First failure is Read Failure.  |
| TG_FIRST_FAIL_IS_WRITE | First failure is Write Failure.   |

*continued...*



| Register Name            | Description  |
|--------------------------|--|
| TG_VERSION               | Reports the traffic generator version number.  |
| TG_NUM_DATA_GEN          | Reports the number of data generators.   |
| TG_NUM_BYTEEN_GEN        | Reports the number of byte-enable generators.  |
| TG_RANK_ADDR_WIDTH       | Reports the rank address width.  |
| TG_BANK_ADDR_WIDTH       | Reports the bank address width.  |
| TG_ROW_ADDR_WIDTH        | Reports the row address width.   |
| TG_BANK_GROUP_WIDTH      | Reports the bank group width.  |
| TG_RDATA_WIDTH           | Reports the width of all data and PNF signals within the traffic generator.                              |
| TG_DATA_PATTERN_LENGTH   | Reports the length of the static pattern to be loaded into static per-pin data generators.               |
| TG_BYTEEN_PATTERN_LENGTH | Reports the length of the static pattern to be loaded into static per-pin byte-enable generators.        |
| TG_MIN_ADDR_INCR         | Reports the minimum address increment permitted for sequential and random-sequential address generation. |
| TG_ERROR_REPORT          | Reports error bits. Refer to <i>Error Report Register Bits</i> for details.                              |
| TG_PNF                   | Read the persistent PNF per bit as an array of 32-bit entries.   |
| TG_FAIL_EXPECTED_DATA    | Reports the first failure expected data. This is read as an array of 32-bit entries.                     |
| TG_FAIL_READ_DATA        | Reports the first failure read data. This is read as an array of 32-bit entries.                         |

The configuration error report register contains information on common misconfigurations of the traffic generator. The bit corresponding to a given configuration error is set high when that configuration error is detected. Ensure that all bits in this register are low, to avoid test failures or unexpected behavior due to improper configuration.

**Table 115. Error Report Register Bits**

| Bit Index (LSB = 0x0) | Bit Name                                | Description of Error  |
|-----------------------|---|---|
| 0                     | ERR_MORE_READS_THAN_WRITES              | You have requested more read transactions per loop than write transactions per loop.  |
| 1                     | ERR_BURSTLENGTH_GT_SEQ_ADDR_I_NCR       | The configured burst length is greater than the configured sequential address increment when address generation is in sequential or random sequential mode.         |
| 2                     | ERR_ADDR_DIVISIBLE_BY_GT_SEQ_A_DDR_INCR | The configured sequential address increment is less than the minimum required address increment when address generation is in sequential or random sequential mode. |

*continued...*



| Bit Index (LSB = 0x0) | Bit Name                                | Description of Error  |
|-----------------------|---|---|
| 3                     | ERR_SEQ_ADDR_INCR_NOT_DIVISIBLE         | The configured sequential address increment is not a multiple of the minimum required address increment when address generation is in sequential or random sequential mode. |
| 4                     | ERR_READ_AND_WRITE_START_ADDRESS_DIFFER | The configured start addresses for reads and writes are different when address generation is in sequential mode.  |
| 5                     | ERR_ADDR_MODES_DIFFERENT                | The configured address modes for reads and writes are different.  |
| 6                     | ERR_NUMBER_OF RAND SEQ_ADDRS DIFFERENT  | The configured number of random sequential addresses for reads and writes are different when address generation is in random sequential mode.                               |
| 7                     | ERR_REPEATS_SET_TO_ZERO                 | The number of read and/or write repeats is set to 0.  |
| 8                     | ERR_BOTH_BURST_AND_REPEAT_MODE_ACTIVE   | The burst length is set greater than 1 and read/write requests are set greater than 1.  |
| 9-31                  | Reserved                                | --  |

### 13.2.2. Performing Your Own Tests Using Traffic Generator 2.0

If you want, you can create your own configuration test stage for the EMIF Configurable Traffic Generator.

The general flow of a configuration test stage, including the default test stages, is as follows:

- Configure the number of loops to be completed by the traffic generator (TG\_LOOP\_COUNT).
- Configure the number of writes and reads to be complete per loop (TG\_WRITE\_COUNT and TG\_READ\_COUNT respectively).
- Choose the burst length of each write and read (TG\_BURST\_LENGTH).
- Select starting write address by writing to the lower and upper bits of the address register (TG\_SEQ\_START\_ADDR\_WR\_L and TG\_SEQ\_START\_ADDR\_WR\_H, respectively).
- Select write address generation mode (TG\_ADDR\_MODE\_WR).
- Select starting read address by writing to the lower and upper bits of the address register (TG\_SEQ\_START\_ADDR\_RD\_L and TG\_SEQ\_START\_ADDR\_RD\_H, respectively).
- Select read address generation mode (TG\_ADDR\_MODE\_RD).
- If applicable, select sequential address increment (TG\_SEQ\_ADDR\_INCR).
- Write initial values/seeds to the data and byte-enable generators (TG\_DATA\_SEED and TG\_BYTEEN\_SEED).
- Select generation mode of the data and byte-enable generators (TG\_DATA\_MODE and TG\_BYTEEN\_MODE).
- Initiate test (TG\_START).



## Simulation

For a comprehensive example of how to write your own configuration test for simulation, refer to the test bench file, located at

<example\_design\_directory>/sim/ed\_sim/  
altera\_emif\_tg\_avl\_2\_>/sim/altera\_emif\_avl\_tg\_2\_tb.sv

To iterate over the data generators or byte-enable generators, you must read the number of data generators and number of byte-enable generators. These values are mapped to read-accessible registers TG\_NUM\_DATA\_GEN and TG\_NUM\_BYTEEN\_GEN, respectively. The following example illustrates how one would configure the data generators to continuously output the pattern *0x5a*, using the simulation test bench:

```
integer num_data_generators;  
...  
    tg_send_cfg_read_0(TG_NUM_DATA_GEN, num_data_generators);  
    tg_send_cfg_write_0(TG_DATA_MODE, 32'h1);  
    for (i = 0; i < num_data_generators; i = i + 1) begin  
        tg_send_cfg_write_0(TG_DATA_SEED + i, 32'h5A);  
    end
```

## Hardware

Configuration test stages in hardware must be inserted into the RTL, and will resemble the single read/write, byte-enable, and block read/write stages in the default test pattern. In most cases, you can modify one of the existing stages to create the desired custom test stage. The stages are linear, finite, state machines that write predetermined values to the configuration address-mapped registers. As always, the last state in configuration is a write to address *0x0* or *TG\_START*. The state machine then waits for the traffic generator to return a signal signifying its completion of the test stage.

Refer to the aforementioned default test stages as examples of hardware test stages. The default test stages are contained within the following files:

- <example\_design\_directory>/qii/ed\_synth/  
altera\_emif\_tg\_avl\_2\_>/synth/altera\_emif\_avl\_tg\_2\_rw\_stage.sv
- <example\_design\_directory>/qii/ed\_synth/  
altera\_emif\_tg\_avl\_2\_>/synth/  
altera\_emif\_avl\_tg\_2\_byteenable\_test\_stage.sv

You can also configure the configurable traffic generator in real time using the EMIF Debug Toolkit. The configuration settings available in the Toolkit interface are detailed in the *Configurable Traffic Generator 2.0 Configuration Options* topic.

### 13.2.3. Signal Splitter Component

The signal splitter (*altera\_emif\_sig\_splitter*) is an internal IP component which receives a single signal as its input and passes that signal directly to *n* outputs, where *n* is an integer value equal to or greater than 1. The signal splitter is useful because Qsys does not directly allow one-to-many connections for conduit interfaces.

The signal splitter contains no logic or memory elements. When you configure the signal splitter to have exactly one output port, it is functionally identical to a single wire, and can be replaced by one with no loss of performance.

The `rzq_splitter` is an instantiation of the signal splitter component specifically for the RZQ signal. The signal splitter facilitates the sharing of one RZQ signal among multiple memory interfaces in an EMIF example design.

## 13.3. UniPHY-Based Example Designs

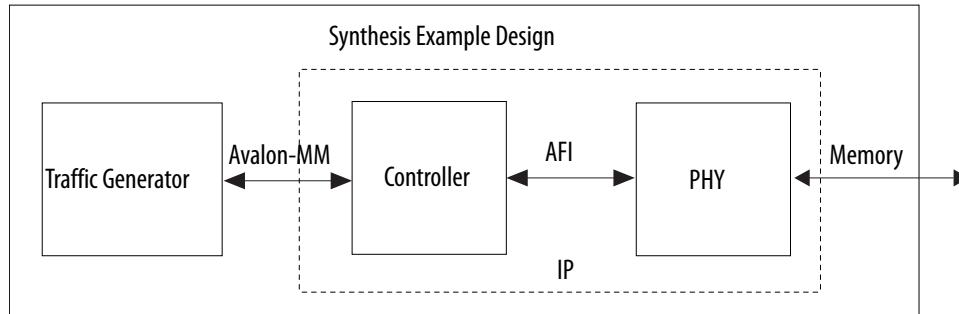
For UniPHY-based interfaces, two independent example designs are created, each containing independent RTL files and other project files. You should compile or simulate these designs separately, and the files should not be mixed. Nonetheless, the designs are related, as the simulation example design builds upon the design of the synthesis example design.

### 13.3.1. Synthesis Example Design

The synthesis example design contains the major blocks shown in the figure below.

- A traffic generator, which is a synthesizable Avalon-MM example driver that implements a pseudo-random pattern of reads and writes to a parameterized number of addresses. The traffic generator also monitors the data read from the memory to ensure it matches the written data and asserts a failure otherwise.
- An instance of the UniPHY memory interface, which includes a memory controller that moderates between the Avalon-MM interface and the AFI interface, and the UniPHY, which serves as an interface between the memory controller and external memory devices to perform read and write operations.

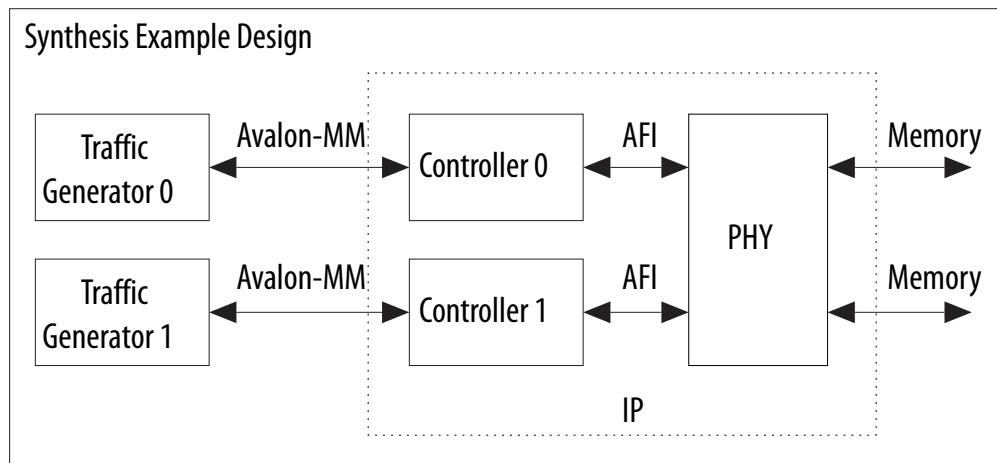
**Figure 176. Synthesis Example Design**



If you are using the Ping Pong PHY feature, the synthesis example design includes two traffic generators issuing commands to two independent memory devices through two independent controllers and a common PHY, as shown in the following figure.

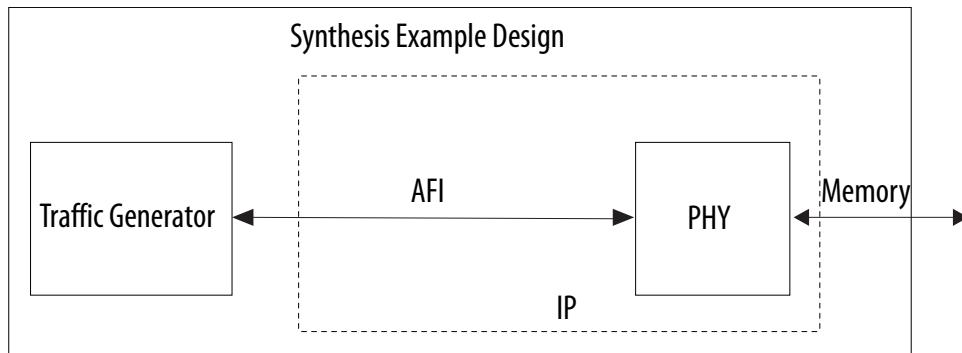


**Figure 177. Synthesis Example Design for Ping Pong PHY**



If you are using RLDRAM 3, the traffic generator in the synthesis example design communicates directly with the PHY using AFI, as shown in the following figure.

**Figure 178. Synthesis Example Design for RLDRAM 3 Interfaces**



You can obtain the synthesis example design by generating your IP core. The files related to the synthesis example design reside at <variation\_name>\_example\_design / example\_project. The synthesis example design includes a Quartus Prime project file (<variation\_name>\_example\_design / example\_project /<variation\_name>.example.qpf). The Quartus Prime project file can be compiled in the Quartus Prime software, and can be run on hardware.

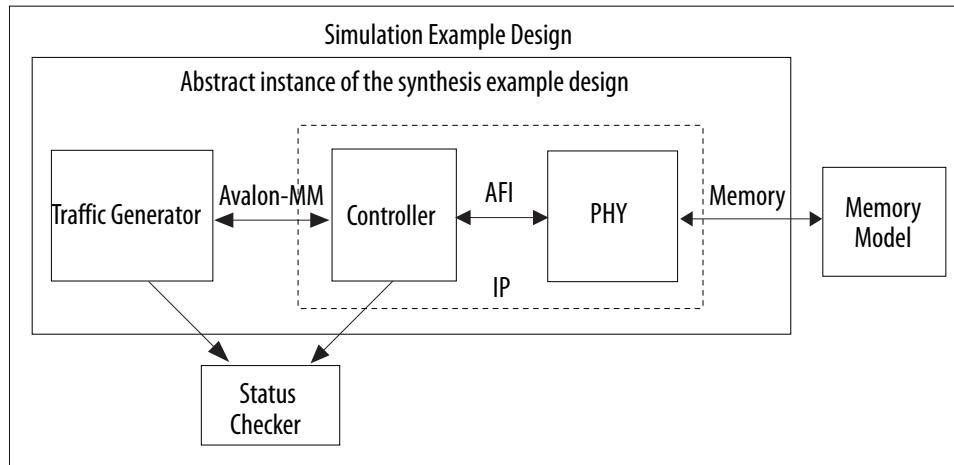
**Note:** If one or more of the **PLL Sharing Mode**, **DLL Sharing Mode**, or **OCT Sharing Mode** parameters are set to any value other than **No Sharing**, the synthesis example design will contain two traffic generator/memory interface instances. The two traffic generator/memory interface instances are related only by shared PLL/DLL/OCT connections as defined by the parameter settings. The traffic generator/memory interface instances demonstrate how you can make such connections in your own designs.

### 13.3.2. Simulation Example Design

The simulation example design contains the major blocks shown in the following figure.

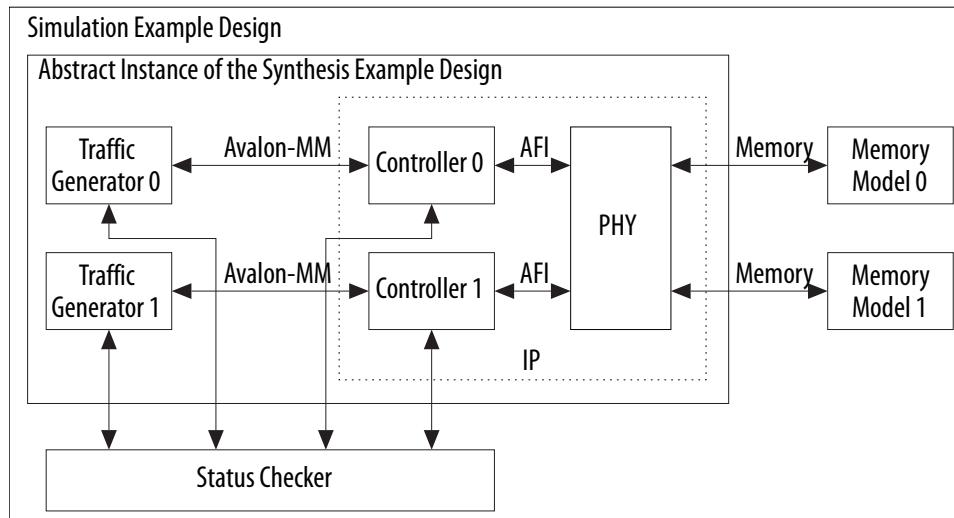
- An instance of the synthesis example design. As described in the previous section, the synthesis example design contains a traffic generator and an instance of the UniPHY memory interface. These blocks default to abstract simulation models where appropriate for rapid simulation.
- A memory model, which acts as a generic model that adheres to the memory protocol specifications. Frequently, memory vendors provide simulation models for specific memory components that you can download from their websites.
- A status checker, which monitors the status signals from the UniPHY IP and the traffic generator, to signal an overall pass or fail condition.

**Figure 179. Simulation Example Design**



If you are using the Ping Pong PHY feature, the simulation example design includes two traffic generators issuing commands to two independent memory devices through two independent controllers and a common PHY, as shown in the following figure.

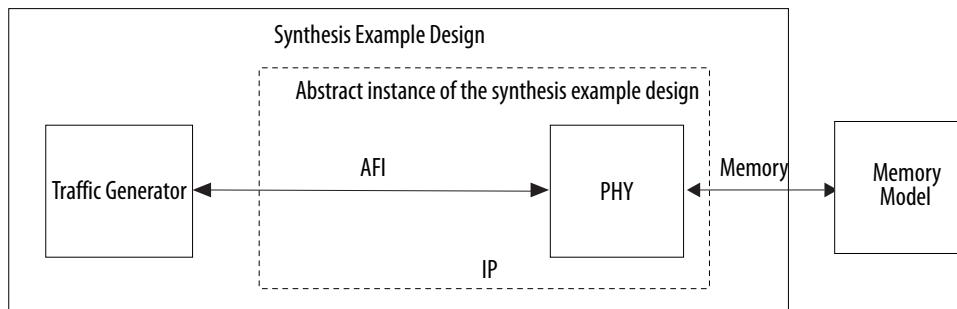
**Figure 180. Simulation Example Design for Ping Pong PHY**



If you are using RLDRAM 3, the traffic generator in the simulation example design communicates directly with the PHY using AFI, as shown in the following figure.



Figure 181. Simulation Example Design for RLDRAM 3 Interfaces

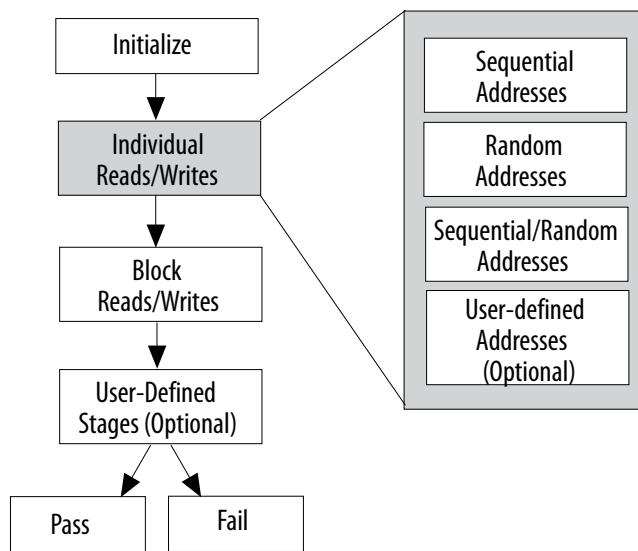


You can obtain the simulation example design by generating your IP core. The files related to the simulation example design reside at <variation\_name>\_example\_design/simulation. After obtaining the generated files, you must still generate the simulation example design RTL for your desired HDL language. The file <variation\_name>\_example\_design/simulation/README.txt contains details about how to generate the IP and to run the simulation in ModelSim or ModelSim - Intel FPGA Edition.

### 13.3.3. Traffic Generator and BIST Engine

The traffic generator and built-in self test (BIST) engine for Avalon-MM memory interfaces generates Avalon-MM traffic on an Avalon-MM master interface. The traffic generator creates read and write traffic, stores the expected read responses internally, and compares the expected responses to the read responses as they arrive. If all reads report their expected response, the pass signal is asserted; however, if any read responds with unexpected data a fail signal occurs.

Each operation generated by the traffic generator is a single write or block of writes followed by a single read or block of reads to the same addresses, which allows the driver to precisely determine the data that should be expected when the read data is returned by the memory interface. The traffic generator comprises a traffic generation block, the Avalon-MM interface and a read comparison block. The traffic generation block generates addresses and write data, which are then sent out over the Avalon-MM interface. The read comparison block compares the read data received from the Avalon-MM interface to the write data from the traffic generator. If at any time the data received is not the expected data, the read comparison block records the failure, finishes reading all the data, and then signals that there is a failure and the traffic generator enters a fail state. If all patterns have been generated and compared successfully, the traffic generator enters a pass state.

**Figure 182. Example Driver Operations**


Within the traffic generator, there are the following main states:

- Generation of individual read and writes
- Generation of block read and writes
- The pass state
- The fail state

Within each of the generation states there are the following substates:

- Sequential address generation
- Random address generation
- Mixed sequential and random address generation

For each of the states and substates, the order and number of operations generated for each substate is parameterizable—you can decide how many of each address pattern to generate, or can disable certain patterns entirely if you want. The sequential and random interleave substate takes in additions to the number of operations to generate. An additional parameter specifies the ratio of sequential to random addresses to generate randomly.

### 13.3.3.1. Read and Write Generation

The traffic generator block can generate individual or block reads and writes.

#### Individual Read and Write Generation

During the traffic generator's individual read and write generation state, the traffic generation block generates individual write followed by individual read Avalon-MM transactions, where the address for the transactions is chosen according to the specific substate. The width of the Avalon-MM interface is a global parameter for the driver, but each substate can have a parameterizable range of burst lengths for each operation.

#### Block Read and Write Generation



During the traffic generator's block read and write generation state, the traffic generator block generates a parameterizable number of write operations followed by the same number of read operations. The specific addresses generated for the blocks are chosen by the specific substates. The burst length of each block operation can be parameterized by a range of acceptable burst lengths.

### 13.3.3.2. Address and Burst Length Generation

The traffic generator block can perform sequential or random addressing.

#### Sequential Addressing

The sequential addressing substate defines a traffic pattern where addresses are chosen in sequential order starting from a user definable address. The number of operations in this substate is parameterizable.

#### Random Addressing

The random addressing substate defines a traffic pattern where addresses are chosen randomly over a parameterizable range. The number of operations in this substate is parameterizable.

#### Sequential and Random Interleaved Addressing

The sequential and random interleaved addressing substate defines a traffic pattern where addresses are chosen to be either sequential or random based on a parameterizable ratio. The acceptable address range is parameterizable as is the number of operations to perform in this substate.

### 13.3.3.3. Traffic Generator Signals

The following table lists the signals used by the traffic generator.

**Table 116. Traffic Generator Signals**

| Signal        | Signal Type   |
|---------------|---|
| clk           | Input clock to traffic generator. For 28nm devices, use the AFI clock. For 20nm devices, use the emif_usr_clk.                          |
| reset_n       | Active low reset input. For 28nm devices, typically connected to afi_reset_n. For 20nm devices, typically connectd to emif_usr_reset_n. |
| avl_ready     | Refer to Avalon Interface Specification.  |
| avl_write_req | Refer to Avalon Interface Specification.  |
| avl_read_req  | Refer to Avalon Interface Specification.  |
| avl_addr      | Refer to Avalon Interface Specification.  |
| avl_size      | Refer to Avalon Interface Specification.  |
| avl_wdata     | Refer to Avalon Interface Specification.  |
| avl_rdata     | Refer to Avalon Interface Specification.  |

*continued...*



| Signal              | Signal Type   |
|---------------------|---|
| avl_rdata_valid     | Refer to Avalon Interface Specification.  |
| pnf_per_bit         | Output. Bitwise pass/fail for last traffic generator read compare of AFI interface. No errors produces value of all 1s.                     |
| pnf_per_bit_persist | Output. Cumulative bitwise pass/fail for all previous traffic generator read compares of AFI interface. No errors produces value of all 1s. |
| pass                | Active high output when traffic generator tests complete successfully.  |
| fail                | Active high output when traffic generator test does not complete successfully.  |
| test_complete       | Active high output when traffic generator test completes.   |

For information about the Avalon signals and the Avalon interface, refer to *Avalon Interface Specifications*.

#### Related Information

[Avalon Interface Specifications](#)

### 13.3.3.4. Traffic Generator Add-Ons

Some optional components that can be useful for verifying aspects of the controller and PHY operation are generated in conjunction with certain user-specified options. These add-on components are self-contained, and are not part of the controller or PHY, nor the traffic generator.

#### User Refresh Generator

The user refresh generator sends refresh requests to the memory controller when user refresh is enabled. The memory controller returns an acknowledgement signal and then issues the refresh command to the memory device.

The user refresh generator is created when you turn on **Enable User Refresh** on the **Controller Settings** tab of the parameter editor.

### 13.3.3.5. Traffic Generator Timeout Counter

The traffic generator timeout counter uses the Avalon interface clock.

When a test fails due to driver failure or timeout, the fail signal is asserted. When a test has failed, the traffic generator must be reset with the `reset_n` signal.

### 13.3.4. Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys

The traffic generator can be used in Qsys as a stand-alone component for use within a larger system.

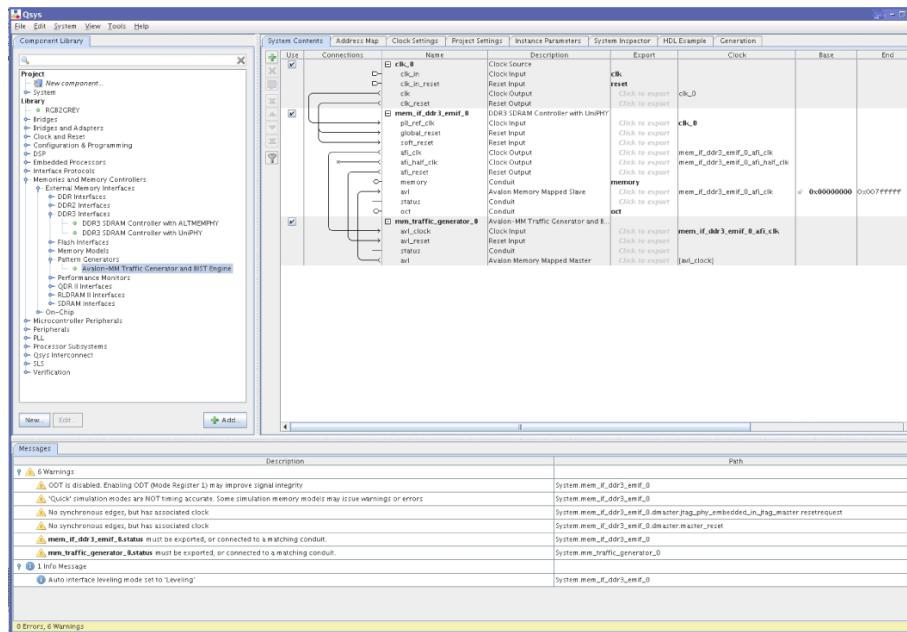
To create the system in Qsys, perform the following steps:

1. Start Qsys.
  2. On the **Project Settings** tab, select the required device from the **Device Family** list.
  3. In the **Component Library**, choose a UniPHY memory interface to instantiate. For example, under **Library > Memories and Memory Controllers > External Memory Interfaces**, select **DDR3 SDRAM Controller with UniPHY**.
  4. Configure the parameters for your instantiation of the memory interface.
  5. In the Component Library, find the example driver and instantiate it in the system. For example, under **Library > Memories and Memory Controllers > Pattern Generators**, select **Avalon-MM Traffic Generator and BIST Engine**.
  6. Configure the parameters for your instantiation of the example driver.

**Note:** The Avalon specification stipulates that Avalon-MM master interfaces issue byte addresses, while Avalon-MM slave interfaces accept word addresses.

The default for the Avalon-MM Traffic Generator and BIST Engine is to issue word addresses. When using Qsys, you must enable the **Generate per byte address** setting in the traffic generator.

7. Connect the interfaces as illustrated in the following figure. At this point, you can generate synthesis RTL, Verilog or VHDL simulation RTL, or a simulation testbench system.



#### **13.3.4.1. Notes on Configuring UniPHY IP in Qsys**

This section includes notes and tips on configuring the UniPHY IP in Qsys.





- The address ranges shown for the Avalon-MM slave interface on the UniPHY component should be interpreted as byte addresses that an Avalon-MM master would address, despite the fact that this range is modified by configuring the word addresses width of the Avalon-MM slave interface on the UniPHY controller.
- The afi\_clk clock source is the associated clock to the Avalon-MM slave interface on the memory controller. This is the ideal clock source to use for all IP components connected on the same Avalon network. Using another clock would cause Qsys to automatically instantiate clock-crossing logic, potentially degrading performance.
- The afi\_clk clock rate is determined by the **Rate on Avalon-MM interface** setting on the UniPHY **PHY Settings** tab. The afi\_half\_clk clock interface has a rate which is further halved. For example, if **Rate on Avalon-MM interface** is set to **Half**, the afi\_clk rate is half of the memory clock frequency, and the afi\_half\_clk is one quarter of the memory clock frequency.
- The global\_reset input interface can be used to reset the UniPHY memory interface and the PLL contained therein. The soft\_reset input interface can be used to reset the UniPHY memory interface but allow the PLL to remain locked. You can use the soft\_reset input to reset the memory but to maintain the AFI clock output to other components in the system.
- Do not connect a reset request from a system component (such as a Nios II processor) to the UniPHY global\_reset\_n port. Doing so would reset the UniPHY PLL, which would propagate as a reset condition on afi\_reset back to the requester; the resulting reset loop could freeze the system.
- Qsys generates an interconnect fabric for each Avalon network. The interconnect fabric is capable of burst and width adaptation. If your UniPHY memory controller is configured with an Avalon interface data width which is wider than an Avalon-MM master interface connected to it, you must enable the byte enable signal on the Avalon-MM slave interface, by checking the **Enable Avalon-MM byte-enable signal** checkbox on the **Controller Settings** tab in the parameter editor.
- If you have a point-to-point connection from an Avalon-MM master to the Avalon-MM slave interface on the memory controller, and if the Avalon data width and burst length settings match, then the Avalon interface data widths may be multiples of either a power of two or nine. Otherwise, you must enable **Generate power-of-2 data bus widths for Qsys or SOPC Builder** on the **Controller Settings** tab of the parameter editor.

## 13.4. Document Revision History

| Date         | Version    | Changes  |
|--------------|------------|--|
| May 2017     | 2017.05.08 | <ul style="list-style-type: none"><li>• Retitled <i>EMIF Configurable Traffic Generator 2.0 Reference to Testing the EMIF Interface Using the Traffic Generator 2.0.</i></li><li>• Removed <i>Configurable Traffic Generator Parameters</i> section.</li><li>• Consolidated Traffic Generator 2.0 usage information in <i>External Memory Interface Debug Toolkit</i> chapter.</li><li>• Rebranded as Intel.</li></ul> |
| October 2016 | 2016.10.31 | Maintenance release.   |
| May 2016     | 2016.05.02 | <ul style="list-style-type: none"><li>• Modified step 6 of <i>Compiling and Testing the Design in Hardware</i>.</li><li>• Added <i>Configuring the Traffic Generator 2.0</i>.</li><li>• Added <i>The Traffic Generator 2.0 Report</i>.</li></ul>   |

*continued...*



| Date          | Version    | Changes  |
|---------------|------------|--|
|               |            | <ul style="list-style-type: none"> <li>Changed heading from <i>EMIF Configurable Traffic Generator 2.0 for Arria 10 EMIF IP</i> to <i>EMIF Configurable Traffic Generator 2.0 Reference</i>.</li> <li>Added <i>Bypass the traffic generator repeated-writes/ repeated-reads test pattern</i>, <i>Bypass the traffic generator stress pattern</i>, and <i>Export Traffic Generator 2.0 configuration interface to Configurable Traffic Generator Parameters table</i>. Removed <i>Number of Traffic Generator 2.0 configuration interfaces</i>.</li> <li>Added <i>TG_WRITE_REPEAT_COUNT</i>, <i>TG_READ_REPEAT_COUNT</i>, <i>TG_DATA_MODE</i>, and <i>TG_BYTEEN_MODE</i> to <i>Configuration Options for Read/Write Generation table</i>.</li> <li>Added <i>Error Report Register Bits table</i> to <i>Test Information section</i>.</li> <li>Corrected paths in <i>Simulation</i> and <i>Hardware</i> sections of <i>Performing Your Own Tests Using Traffic Generator 2.0</i> topic.</li> </ul> |
| November 2015 | 2015.11.02 | <ul style="list-style-type: none"> <li>Added <i>EMIF Configurable Traffic Generator 2.0 for Arria 10 EMIF IP</i>.</li> <li>Added <i>Arria 10 EMIF IP Example Design Quick Start Guide</i>.</li> <li>Changed order of sections in chapter.</li> <li>Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i>.</li> </ul>  |
| May 2015      | 2015.05.04 | Maintenance release.   |
| December 2014 | 2014.12.15 | Maintenance release.   |
| August 2014   | 2014.08.15 | Removed occurrence of <i>MegaWizard Plug-In Manager</i> .  |
| December 2013 | 2013.12.16 | Removed references to <i>SOPC Builder</i> .  |
| November 2012 | 1.3        | <ul style="list-style-type: none"> <li>Added block diagrams of simulation and synthesis example designs for <i>RLDRAM 3</i> and <i>Ping Pong PHY</i>.</li> <li>Changed chapter number from 7 to 9.</li> </ul>  |
| June 2012     | 1.2        | Added Feedback icon.   |
| November 2011 | 1.1        | <ul style="list-style-type: none"> <li>Added <i>Synthesis Example Design</i> and <i>Simulation Example Design</i> sections.</li> <li>Added <i>Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys</i>.</li> <li>Revised Example Driver section as <i>Traffic Generator and BIST Engine</i>.</li> </ul>   |



## 14. Introduction to UniPHY IP

The UniPHY IP is an interface between a memory controller and memory devices and performs read and write operations to the memory. The UniPHY IP creates the datapath between the memory device and the memory controller and user logic in various Intel devices.

The Intel FPGA DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY, QDR II and QDR II+ SRAM controllers with UniPHY, RLDRAM II controller with UniPHY, and RLDRAM 3 PHY-only IP provide low latency, high-performance, feature-rich interfaces to industry-standard memory devices. The DDR2, QDR II and QDR II+, and RLDRAM II controllers with UniPHY offer full-rate and half-rate interfaces, while the DDR3 controller with UniPHY and the RLDRAM 3 PHY-only IP offer half-rate and quarter-rate interfaces, and the LPDDR2 controller with UniPHY offers a half-rate interface.

When you generate your external memory interface IP core, the system creates an example top-level project, consisting of an example driver, and your controller custom variation. The controller instantiates an instance of the UniPHY datapath.

The example top-level project is a fully-functional design that you can simulate, synthesize, and use in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass, fail, and test-complete signals.

If the UniPHY datapath does not match your requirements, you can create your own memory interface datapath using the ALTDLL, ALTDQ\_DQS, ALTDQ\_DQS2, ALTDQ, or ALTDQS IP cores, available in the Quartus Prime software, but you are then responsible for all aspects of the design including timing analysis and design constraints.

### 14.1. Release Information

The following table provides information about this release of the DDR2 and DDR3 SDRAM, QDR II and QDR II+ SRAM, and RLDRAM II controllers with UniPHY, and the RLDRAM 3 PHY-only IP.

**Table 117. Release Information**

| Item          | Protocol                                      |               |               |               |
|---------------|---|---------------|---------------|---------------|
|               | DDR2, DDR3,<br>LPDDR2                         | QDR II        | RLDRAM II     | RLDRAM 3      |
| Version       | 13.1  | 13.1          | 13.1          | 13.1          |
| Release Date  | November 2013                                 | November 2013 | November 2013 | November 2013 |
| Ordering Code | IP-DDR2/UNI<br>IP-DDR3/UNI<br>IP-SDRAM/LPDDR2 | IP-QDRII/UNI  | IP-RLDII/UNI  | —             |

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Empirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.

ISO  
9001:2015  
Registered



Intel verifies that the current version of the Quartus Prime software compiles the previous version of each IP core. The *Intel FPGA IP Library Release Notes and Errata* report any exceptions to this verification. Intel does not verify compilation with IP core versions older than one release.

#### Related Information

[Intel FPGA IP Library Release Notes](#)

## 14.2. Device Support Levels

The following terms define the device support levels for Intel FPGA IP cores.

#### Intel FPGA IP Core Device Support Levels

- **Preliminary support**—Intel verifies the IP core with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. You can use it in production designs with caution.
- **Final support**—Intel verifies the IP core with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

## 14.3. Device Family and Protocol Support

The following table shows the level of support offered by each of the UniPHY-based external memory interface protocols for Intel device families.

**Table 118. Device Family Support**

| Device Family         | Support Level   |                                   |            |   |                                   |            |
|-----------------------|---|-----------------------------------|------------|---|-----------------------------------|------------|
|                       | DDR2  | DDR3                              | LPDDR2     | QDR II  | RDRAM II                          | RDRAM 3    |
| Arria® II GX          | No support  | No support                        | No support | Final   | No support                        | No support |
| Arria II GZ           | Final   | Final                             | No support | Final   | Final                             | No support |
| Arria V               | Refer to the <i>What's New in Intel FPGA IP</i> on <a href="http://www.altera.com">www.altera.com</a> . |                                   |            |   |                                   |            |
| Arria V GZ            | Refer to the <i>What's New in Intel FPGA IP</i> on <a href="http://www.altera.com">www.altera.com</a> . |                                   | No support | Refer to the <i>What's New in Intel FPGA IP</i> on <a href="http://www.altera.com">www.altera.com</a> . |                                   |            |
| Cyclone V             | Refer to the <i>What's New in Intel FPGA IP</i> on <a href="http://www.altera.com">www.altera.com</a> . |                                   |            |   |                                   | No support |
| Stratix® III          | Final   | Final (Only Vcc = 1.1V supported) | No support | Final   | Final (Only Vcc = 1.1V supported) | No support |
| Stratix IV            | Final   | Final                             | No support | Final   | Final                             | No support |
| Stratix V             | Refer to the <i>What's New in Intel FPGA IP</i> on <a href="http://www.altera.com">www.altera.com</a> . |                                   | No support | Refer to the <i>What's New in Intel FPGA IP</i> on <a href="http://www.altera.com">www.altera.com</a> . |                                   |            |
| Other device families | No support  | No support                        | No support | No support  | No support                        | No support |

For information about features and supported clock rates for external memory interfaces, refer to the *External Memory Specification Estimator*.

### Related Information

- What's New in Intel FPGA IP
- External Memory Interface Spec Estimator

## 14.4. UniPHY-Based External Memory Interface Features

The following table summarizes key feature support for Intel's UniPHY-based external memory interfaces.

**Table 119. Feature Support**

| Key Feature   | Protocol               |              |         |                                 |             |            |
|---|------------------------|--------------|---------|---------------------------------|-------------|------------|
|   | DDR2                   | DDR3         | LPDDR2  | QDR II                          | RDRAM II    | RDRAM 3    |
| High-performance controller II (HPC II)                               | Yes                    | Yes          | Yes     | —                               | —           | —          |
| Half-rate core logic and user interface                               | Yes                    | Yes          | Yes     | Yes                             | Yes         | Yes        |
| Full-rate core logic and user interface                               | Yes                    | —            | —       | Yes                             | Yes         | —          |
| Quarter-rate core logic and user interface                            | —                      | Yes (1)      | —       | —                               | —           | Yes        |
| Dynamically generated Nios II-based sequencer                         | Yes                    | Yes          | Yes     | Yes                             | Yes         | Yes        |
| Choice of RTL-based or dynamically generated Nios® II-based sequencer | —                      | —            | —       | Yes (2) (3)<br>(12)             | Yes (12)    | —          |
| Available Efficiency Monitor and Protocol Checker (14)                | Yes                    | Yes          | Yes     | —                               | Yes         | Yes        |
| DDR3L support   | —                      | Yes (13)     | —       | —                               | —           | —          |
| UDIMM and RDIMM in any form factor                                    | Yes                    | Yes (4) (5)  | —       | —                               | —           | —          |
| Multiple components in a single-rank UDIMM or RDIMM layout            | Yes                    | Yes          | —       | —                               | —           | —          |
| LRDIMM  | —                      | Yes          | —       | —                               | —           | —          |
| Burst length (half-rate)  | 8                      | —            | 8 or 16 | 4                               | 4 or 8      | 2, 4, or 8 |
| Burst length (full-rate)  | 4                      | —            | —       | 2 or 4                          | 2, 4, or 8  | —          |
| Burst length (quarter-rate)   | —                      | 8            | —       | —                               | —           | 2, 4, or 8 |
| Burst length of 8 and burst chop of 4 (on the fly)                    | —                      | Yes          | —       | —                               | —           | —          |
| With leveling   | 240 MHz and above (10) | Yes (9) (10) | —       | —                               | —           | Yes        |
| Without leveling  | Below 240 MHz          | —            | Yes     | —                               | —           | —          |
| Maximum data width  | 144 bits (6)           | 144 bits (6) | 32 bits | 72 bits                         | 72 bits     | 72 bits    |
| Reduced controller latency  | —                      | —            | —       | Yes (2) (7)                     | Yes (2) (7) | —          |
| Read latency  | —                      | —            | —       | 1.5 (QDR II) 2 or 2.5 (QDR II+) | —           | —          |

*continued...*



| Key Feature            | Protocol |      |        |              |           |          |
|------------------------|----------|------|--------|--------------|-----------|----------|
|                        | DDR2     | DDR3 | LPDDR2 | QDR II       | RLDRAM II | RLDRAM 3 |
| ODT (in memory device) | —        | Yes  | —      | QDR II+ only | Yes       | Yes      |
| x36 emulation mode     | —        | —    | —      | Yes (8) (10) | —         | —        |

Notes:

1. For Arria V, Arria V GZ, and Stratix V devices only.
2. Not available in Arria II GX devices.
3. Nios II-based sequencer not available for full-rate interfaces.
4. For DDR3, the DIMM form is not supported in Arria II GX, Arria II GZ, Arria V, or Cyclone V devices.
5. Arria II GZ uses leveling logic for discrete devices in DDR3 interfaces to achieve high speeds, but that leveling cannot be used to implement the DIMM form in DDR3 interfaces.
6. For any interface with data width above 72 bits, you must use software timing analysis of your complete design to determine the maximum clock rate.
7. The maximum achievable clock rate when reduced controller latency is selected must be attained through Quartus Prime software timing analysis of your complete design.
8. Emulation mode allows emulation of a larger memory-width interface using multiple smaller memory-width interfaces. For example, an x36 QDR II or QDR II+ interface can be emulated using two x18 interfaces.
9. The leveling delay on the board between first and last DDR3 SDRAM component laid out as a DIMM must be less than 0.69 tCK.
10. Leveling is not available for Arria V or Cyclone V devices.
11. x36 emulation mode is not supported in Arria V, Arria V GZ, Cyclone V, or Stratix V devices.
12. The RTL-based sequencer is not available for QDR II or RLDRAM II interfaces on Arria V devices.
13. For Arria V, Arria V GZ, Cyclone V, and Stratix V.
14. The Efficiency Monitor and Protocol Checker is not available for QDR II and QDR II+ SRAM, or for the MAX 10 device family, or for Arria V or Cyclone V designs using the Hard Memory Controller.

## 14.5. System Requirements

For system requirements and installation instructions, refer to *Intel FPGA Software Installation and Licensing* manual.

The DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY, QDR II and QDR II+ SRAM controllers with UniPHY, RLDRAM II controller with UniPHY, and RLDRAM 3 PHY-only IP are part of the Intel FPGA IP Library, which Intel distributes with the Quartus Prime software.

### Related Information

[Intel FPGA Software Installation and Licensing Manual](#)

## 14.6. Intel FPGA IP Core Verification

Intel has carried out extensive random, directed tests with functional test coverage using industry-standard models to ensure the functionality of the external memory controllers with UniPHY. Intel's functional verification of the external memory controllers with UniPHY use modified Denali models, with certain assertions disabled.

## 14.7. Resource Utilization

The following topics provide resource utilization data for the external memory controllers with UniPHY for supported device families.

### 14.7.1. DDR2, DDR3, and LPDDR2 Resource Utilization in Arria V Devices

The following table shows typical resource usage of the DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY in the current version of Quartus Prime software for Arria V devices.

**Table 120. Resource Utilization in Arria V Devices**

| Protocol           | Memory Width (Bits) | Combinational ALUTS | Logic Registers | M10K Blocks | Memory (Bits) | Hard Memory Controller |
|--------------------|---------------------|---------------------|-----------------|-------------|---------------|------------------------|
| <b>Controller</b>  |                     |                     |                 |             |               |                        |
| DDR2 (Half rate)   | 8                   | 2286                | 1404            | 4           | 6560          | 0                      |
|                    | 64                  | 2304                | 1379            | 17          | 51360         | 0                      |
| DDR2 (Fullrate)    | 32                  | 0                   | 0               | 0           | 0             | 1                      |
| DDR3 (Half rate)   | 8                   | 2355                | 1412            | 4           | 6560          | 0                      |
|                    | 64                  | 2372                | 1440            | 17          | 51360         | 0                      |
| DDR3 (Full rate)   | 32                  | 0                   | 0               | 0           | 0             | 1                      |
| LPDDR2 (Half rate) | 8                   | 2230                | 1617            | 4           | 6560          | 0                      |
|                    | 32                  | 2239                | 1600            | 10          | 25760         | 0                      |
| <b>PHY</b>         |                     |                     |                 |             |               |                        |
| DDR2 (Half rate)   | 8                   | 1652                | 2015            | 34          | 141312        | 0                      |
|                    | 64                  | 1819                | 2089            | 34          | 174080        | 0                      |
| DDR2 (Fullrate)    | 32                  | 1222                | 1415            | 34          | 157696        | 1                      |
| DDR3 (Half rate)   | 8                   | 1653                | 1977            | 34          | 141312        | 0                      |
|                    | 64                  | 1822                | 2090            | 34          | 174080        | 0                      |
| DDR3 (Full rate)   | 32                  | 1220                | 1428            | 34          | 157696        | 0                      |
| LPDDR2 (Half rate) | 8                   | 2998                | 3187            | 35          | 150016        | 0                      |
|                    | 32                  | 3289                | 3306            | 35          | 174592        | 0                      |
| <b>Total</b>       |                     |                     |                 |             |               |                        |
| DDR2 (Half rate)   | 8                   | 4555                | 3959            | 39          | 148384        | 0                      |
|                    | 64                  | 4991                | 4002            | 52          | 225952        | 0                      |
| DDR2 (Fullrate)    | 32                  | 1776                | 1890            | 35          | 158208        | 1                      |
| DDR3 (Half rate)   | 8                   | 4640                | 3934            | 39          | 148384        | 0                      |
|                    | 64                  | 5078                | 4072            | 52          | 225952        | 0                      |
| DDR3 (Full rate)   | 32                  | 1774                | 1917            | 35          | 158208        | 1                      |
| LPDDR2 (Half rate) | 8                   | 5228                | 4804            | 39          | 156576        | 0                      |
|                    | 32                  | 5528                | 4906            | 45          | 200352        | 0                      |



## 14.7.2. DDR2 and DDR3 Resource Utilization in Arria II GZ Devices

The following table shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus Prime software for Arria II GZ devices.

**Table 121. Resource Utilization in Arria II GZ Devices**

| Protocol          | Memory Width (Bits) | Combination al ALUTs | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|-------------------|---------------------|----------------------|-----------------|-----------|------------|--------------|---------------|
| <b>Controller</b> |                     |                      |                 |           |            |              |               |
| DDR2 (Half rate)  | 8                   | 1,781                | 1,092           | 10        | 2          | 0            | 4,352         |
|                   | 16                  | 1,784                | 1,092           | 10        | 4          | 0            | 8,704         |
|                   | 64                  | 1,818                | 1,108           | 10        | 15         | 0            | 34,560        |
|                   | 72                  | 1,872                | 1,092           | 10        | 17         | 0            | 39,168        |
| DDR2 (Full rate)  | 8                   | 1,851                | 1,124           | 10        | 2          | 0            | 2,176         |
|                   | 16                  | 1,847                | 1,124           | 10        | 2          | 0            | 4,352         |
|                   | 64                  | 1,848                | 1,124           | 10        | 8          | 0            | 17,408        |
|                   | 72                  | 1,852                | 1,124           | 10        | 9          | 0            | 19,574        |
| DDR3 (Half rate)  | 8                   | 1,869                | 1,115           | 10        | 2          | 0            | 4,352         |
|                   | 16                  | 1,868                | 1,115           | 10        | 4          | 0            | 8,704         |
|                   | 64                  | 1,882                | 1,131           | 10        | 15         | 0            | 34,560        |
|                   | 72                  | 1,888                | 1,115           | 10        | 17         | 0            | 39,168        |
| <b>PHY</b>        |                     |                      |                 |           |            |              |               |
| DDR2 (Half rate)  | 8                   | 2,560                | 2,042           | 183       | 22         | 0            | 157,696       |
|                   | 16                  | 2,730                | 2,262           | 183       | 22         | 0            | 157,696       |
|                   | 64                  | 3,606                | 3,581           | 183       | 22         | 0            | 157,696       |
|                   | 72                  | 3,743                | 3,796           | 183       | 22         | 0            | 157,696       |
| DDR2 (Full rate)  | 8                   | 2,494                | 1,934           | 169       | 22         | 0            | 157,696       |
|                   | 16                  | 2,652                | 2,149           | 169       | 22         | 0            | 157,696       |
|                   | 64                  | 3,519                | 3,428           | 169       | 22         | 0            | 157,696       |
|                   | 72                  | 3,646                | 3,642           | 169       | 22         | 0            | 157,696       |
| DDR3 (Half rate)  | 8                   | 2,555                | 2,032           | 187       | 22         | 0            | 157,696       |
|                   | 16                  | 3,731                | 2,251           | 187       | 22         | 0            | 157,696       |
|                   | 64                  | 3,607                | 3,572           | 187       | 22         | 0            | 157,696       |
|                   | 72                  | 3,749                | 3,788           | 187       | 22         | 0            | 157,696       |
| <b>Total</b>      |                     |                      |                 |           |            |              |               |
| DDR2 (Half rate)  | 8                   | 4,341                | 3,134           | 193       | 24         | 0            | 4,374         |
|                   | 16                  | 4,514                | 3,354           | 193       | 26         | 0            | 166,400       |
|                   | 64                  | 5,424                | 4,689           | 193       | 37         | 0            | 192,256       |

*continued...*



| Protocol         | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|------------------|---------------------|---------------------|-----------------|-----------|------------|--------------|---------------|
|                  | 72                  | 5,615               | 4,888           | 193       | 39         | 0            | 196,864       |
| DDR2 (Full rate) | 8                   | 4,345               | 3,058           | 179       | 24         | 0            | 159,872       |
|                  | 16                  | 4,499               | 3,273           | 179       | 24         | 0            | 162,048       |
|                  | 64                  | 5,367               | 4,552           | 179       | 30         | 0            | 175,104       |
|                  | 72                  | 5,498               | 4,766           | 179       | 31         | 0            | 177,280       |
| DDR3 (Half rate) | 8                   | 4,424               | 3,147           | 197       | 24         | 0            | 162,048       |
|                  | 16                  | 5,599               | 3,366           | 197       | 26         | 0            | 166,400       |
|                  | 64                  | 5,489               | 4,703           | 197       | 37         | 0            | 192,256       |
|                  | 72                  | 5,637               | 4,903           | 197       | 39         | 0            | 196,864       |

### 14.7.3. DDR2 and DDR3 Resource Utilization in Stratix III Devices

The following table shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus Prime software for Stratix III devices.

**Table 122. Resource Utilization in Stratix III Devices**

| Protocol          | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|-------------------|---------------------|---------------------|-----------------|-----------|------------|--------------|---------------|
| <b>Controller</b> |                     |                     |                 |           |            |              |               |
| DDR2 (Half rate)  | 8                   | 1,807               | 1,058           | 0         | 4          | 0            | 4,464         |
|                   | 16                  | 1,809               | 1,058           | 0         | 6          | 0            | 8,816         |
|                   | 64                  | 1,810               | 1,272           | 10        | 14         | 0            | 32,256        |
|                   | 72                  | 1,842               | 1,090           | 10        | 17         | 0            | 39,168        |
| DDR2 (Full rate)  | 8                   | 1,856               | 1,093           | 0         | 4          | 0            | 2,288         |
|                   | 16                  | 1,855               | 1,092           | 0         | 4          | 0            | 4,464         |
|                   | 64                  | 1,841               | 1,092           | 0         | 10         | 0            | 17,520        |
|                   | 72                  | 1,834               | 1,092           | 0         | 11         | 0            | 19,696        |
| DDR3 (Half rate)  | 8                   | 1,861               | 1,083           | 0         | 4          | 0            | 4,464         |
|                   | 16                  | 1,863               | 1,083           | 0         | 6          | 0            | 8,816         |
|                   | 64                  | 1,878               | 1,295           | 10        | 14         | 0            | 32,256        |
|                   | 72                  | 1,895               | 1,115           | 10        | 17         | 0            | 39,168        |
| <b>PHY</b>        |                     |                     |                 |           |            |              |               |
| DDR2 (Half rate)  | 8                   | 2,591               | 2,100           | 218       | 6          | 1            | 157,696       |
|                   | 16                  | 2,762               | 2,320           | 218       | 6          | 1            | 157,696       |
|                   | 64                  | 3,672               | 3,658           | 242       | 6          | 1            | 157,696       |
|                   | 72                  | 3,814               | 3,877           | 242       | 6          | 1            | 157,696       |
| DDR2 (Full rate)  | 8                   | 2,510               | 1,986           | 200       | 6          | 1            | 157,696       |

*continued...*



| Protocol         | Memory Width (Bits) | Combination al ALUTs | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|------------------|---------------------|----------------------|-----------------|-----------|------------|--------------|---------------|
|                  | 16                  | 2,666                | 2,200           | 200       | 6          | 1            | 157,696       |
|                  | 64                  | 3,571                | 3,504           | 224       | 6          | 1            | 157,696       |
|                  | 72                  | 3,731                | 3,715           | 224       | 6          | 1            | 157,696       |
| DDR3 (Half rate) | 8                   | 2,591                | 2,094           | 224       | 6          | 1            | 157,696       |
|                  | 16                  | 2,765                | 2,314           | 224       | 6          | 1            | 157,696       |
|                  | 64                  | 3,680                | 3,653           | 248       | 6          | 1            | 157,696       |
|                  | 72                  | 3,819                | 3,871           | 248       | 6          | 1            | 157,696       |
| <b>Total</b>     |                     |                      |                 |           |            |              |               |
| DDR2 (Half rate) | 8                   | 4,398                | 3,158           | 218       | 10         | 1            | 162,160       |
|                  | 16                  | 4,571                | 3,378           | 218       | 12         | 1            | 166,512       |
|                  | 64                  | 5,482                | 4,930           | 252       | 20         | 1            | 189,952       |
|                  | 72                  | 5,656                | 4,967           | 252       | 23         | 1            | 196,864       |
| DDR2 (Full rate) | 8                   | 4,366                | 3,079           | 200       | 10         | 1            | 159,984       |
|                  | 16                  | 4,521                | 3,292           | 200       | 10         | 1            | 162,160       |
|                  | 64                  | 5,412                | 4,596           | 224       | 16         | 1            | 175,216       |
|                  | 72                  | 5,565                | 4,807           | 224       | 17         | 1            | 177,392       |
| DDR3 (Half rate) | 8                   | 4,452                | 3,177           | 224       | 10         | 1            | 162,160       |
|                  | 16                  | 4,628                | 3,397           | 224       | 12         | 1            | 166,512       |
|                  | 64                  | 5,558                | 4,948           | 258       | 20         | 1            | 189,952       |
|                  | 72                  | 5,714                | 4,986           | 258       | 23         | 1            | 196,864       |

#### 14.7.4. DDR2 and DDR3 Resource Utilization in Stratix IV Devices

The following table shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus Prime software for Stratix IV devices.

**Table 123. Resource Utilization in Stratix IV Devices**

| Protocol          | Memory Width (Bits) | Combination al ALUTs | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|-------------------|---------------------|----------------------|-----------------|-----------|------------|--------------|---------------|
| <b>Controller</b> |                     |                      |                 |           |            |              |               |
| DDR2 (Half rate)  | 8                   | 1,785                | 1,090           | 10        | 2          | 0            | 4,352         |
|                   | 16                  | 1,785                | 1,090           | 10        | 4          | 0            | 8,704         |
|                   | 64                  | 1,796                | 1,106           | 10        | 15         | 0            | 34,560        |
|                   | 72                  | 1,798                | 1,090           | 10        | 17         | 0            | 39,168        |
| DDR2 (Full rate)  | 8                   | 1,843                | 1,124           | 10        | 2          | 0            | 2,176         |
|                   | 16                  | 1,845                | 1,124           | 10        | 2          | 0            | 4,352         |
|                   | 64                  | 1,832                | 1,124           | 10        | 8          | 0            | 17,408        |

*continued...*



| Protocol         | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Mem ALUTs | M9K Blocks | M144K Blocks | Memory (Bits) |
|------------------|---------------------|---------------------|-----------------|-----------|------------|--------------|---------------|
|                  | 72                  | 1,834               | 1,124           | 10        | 9          | 0            | 19,584        |
| DDR3 (Half rate) | 8                   | 1,862               | 1,115           | 10        | 2          | 0            | 4,352         |
|                  | 16                  | 1,874               | 1,115           | 10        | 4          | 0            | 8,704         |
|                  | 64                  | 1,880               | 1,131           | 10        | 15         | 0            | 34,560        |
|                  | 72                  | 1,886               | 1,115           | 10        | 17         | 0            | 39,168        |
| <b>PHY</b>       |                     |                     |                 |           |            |              |               |
| DDR2 (Half rate) | 8                   | 2,558               | 2,041           | 183       | 6          | 1            | 157,696       |
|                  | 16                  | 2,728               | 2,262           | 183       | 6          | 1            | 157,696       |
|                  | 64                  | 3,606               | 3,581           | 183       | 6          | 1            | 157,696       |
|                  | 72                  | 3,748               | 3,800           | 183       | 6          | 1            | 157,696       |
| DDR2 (Full rate) | 8                   | 2,492               | 1,934           | 169       | 6          | 1            | 157,696       |
|                  | 16                  | 2,652               | 2,148           | 169       | 6          | 1            | 157,696       |
|                  | 64                  | 3,522               | 3,428           | 169       | 6          | 1            | 157,696       |
|                  | 72                  | 3,646               | 3,641           | 169       | 6          | 1            | 157,696       |
| DDR3 (Half rate) | 8                   | 2,575               | 2,031           | 187       | 6          | 1            | 157,696       |
|                  | 16                  | 2,732               | 2,251           | 187       | 6          | 1            | 157,696       |
|                  | 64                  | 3,602               | 3,568           | 187       | 6          | 1            | 157,696       |
|                  | 72                  | 3,750               | 3,791           | 187       | 6          | 1            | 157,696       |
| <b>Total</b>     |                     |                     |                 |           |            |              |               |
| DDR2 (Half rate) | 8                   | 4,343               | 3,131           | 193       | 8          | 1            | 162,048       |
|                  | 16                  | 4,513               | 3,352           | 193       | 10         | 1            | 166,400       |
|                  | 64                  | 5,402               | 4,687           | 193       | 21         | 1            | 192,256       |
|                  | 72                  | 5,546               | 4,890           | 193       | 23         | 1            | 196,864       |
| DDR2 (Full rate) | 8                   | 4,335               | 3,058           | 179       | 8          | 1            | 159,872       |
|                  | 16                  | 4,497               | 3,272           | 179       | 8          | 1            | 162,048       |
|                  | 64                  | 5,354               | 4,552           | 179       | 14         | 1            | 175,104       |
|                  | 72                  | 5,480               | 4,765           | 179       | 15         | 1            | 177,280       |
| DDR3 (Half rate) | 8                   | 4,437               | 3,146           | 197       | 8          | 1            | 162,048       |
|                  | 16                  | 4,606               | 3,366           | 197       | 10         | 1            | 166,400       |
|                  | 64                  | 5,482               | 4,699           | 197       | 21         | 1            | 192,256       |
|                  | 72                  | 5,636               | 4,906           | 197       | 23         | 1            | 196,864       |

#### 14.7.5. DDR2 and DDR3 Resource Utilization in Arria V GZ and Stratix V Devices

The following table shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus Prime software for Arria V GZ and Stratix V devices.



**Table 124. Resource Utilization in Arria V GZ and Stratix V Devices**

| Protocol            | Memory Width (Bits) | Combinational LCs | Logic Registers | M20K Blocks | Memory (Bits) |
|---------------------|---------------------|-------------------|-----------------|-------------|---------------|
| <b>Controller</b>   |                     |                   |                 |             |               |
| DDR2 (Half rate)    | 8                   | 1,787             | 1,064           | 2           | 4,352         |
|                     | 16                  | 1,794             | 1,064           | 4           | 8,704         |
|                     | 64                  | 1,830             | 1,070           | 14          | 34,304        |
|                     | 72                  | 1,828             | 1,076           | 15          | 38,400        |
| DDR2 (Full rate)    | 8                   | 2,099             | 1,290           | 2           | 2,176         |
|                     | 16                  | 2,099             | 1,290           | 2           | 4,352         |
|                     | 64                  | 2,126             | 1,296           | 7           | 16,896        |
|                     | 72                  | 2,117             | 1,296           | 8           | 19,456        |
| DDR3 (Quarter rate) | 8                   | 2,101             | 1,370           | 4           | 8,704         |
|                     | 16                  | 2,123             | 1,440           | 7           | 16,896        |
|                     | 64                  | 2,236             | 1,885           | 28          | 69,632        |
|                     | 72                  | 2,102             | 1,870           | 30          | 74,880        |
| DDR3 (Half rate)    | 8                   | 1,849             | 1,104           | 2           | 4,352         |
|                     | 16                  | 1,851             | 1,104           | 4           | 8,704         |
|                     | 64                  | 1,853             | 1,112           | 14          | 34,304        |
|                     | 72                  | 1,889             | 1,116           | 15          | 38,400        |
| <b>PHY</b>          |                     |                   |                 |             |               |
| DDR2 (Half rate)    | 8                   | 2,567             | 1,757           | 13          | 157,696       |
|                     | 16                  | 2,688             | 1,809           | 13          | 157,696       |
|                     | 64                  | 3,273             | 2,115           | 13          | 157,696       |
|                     | 72                  | 3,377             | 2,166           | 13          | 157,696       |
| DDR2 (Full rate)    | 8                   | 2,491             | 1,695           | 13          | 157,696       |
|                     | 16                  | 2,578             | 1,759           | 13          | 157,696       |
|                     | 64                  | 3,062             | 2,137           | 13          | 157,696       |
|                     | 72                  | 3,114             | 2,200           | 13          | 157,696       |
| DDR3 (Quarter rate) | 8                   | 2,209             | 2,918           | 18          | 149,504       |
|                     | 16                  | 2,355             | 3,327           | 18          | 157,696       |
|                     | 64                  | 3,358             | 5,228           | 18          | 182,272       |
|                     | 72                  | 4,016             | 6,318           | 18          | 198,656       |
| DDR3 (Half rate)    | 8                   | 2,573             | 1,791           | 13          | 157,696       |
|                     | 16                  | 2,691             | 1,843           | 13          | 157,696       |
|                     | 64                  | 3,284             | 2,149           | 13          | 157,696       |
|                     | 72                  | 3,378             | 2,200           | 13          | 157,696       |

*continued...*



| Protocol            | Memory Width (Bits) | Combinational LCs | Logic Registers | M20K Blocks | Memory (Bits) |
|---------------------|---------------------|-------------------|-----------------|-------------|---------------|
| <b>Total</b>        |                     |                   |                 |             |               |
| DDR2 (Half rate)    | 8                   | 4,354             | 2,821           | 15          | 162,048       |
|                     | 16                  | 4,482             | 2,873           | 17          | 166,400       |
|                     | 64                  | 5,103             | 3,185           | 27          | 192,000       |
|                     | 72                  | 5,205             | 3,242           | 28          | 196,096       |
| DDR2 (Full rate)    | 8                   | 4,590             | 2,985           | 15          | 159,872       |
|                     | 16                  | 4,677             | 3,049           | 15          | 162,048       |
|                     | 64                  | 5,188             | 3,433           | 20          | 174,592       |
|                     | 72                  | 5,231             | 3,496           | 21          | 177,152       |
| DDR3 (Quarter rate) | 8                   | 4,897             | 4,844           | 23          | 158,720       |
|                     | 16                  | 5,065             | 5,318           | 26          | 175,104       |
|                     | 64                  | 6,183             | 7,669           | 47          | 252,416       |
|                     | 72                  | 6,705             | 8,744           | 49          | 274,048       |
| DDR3 (Half rate)    | 8                   | 4,422             | 2,895           | 15          | 162,048       |
|                     | 16                  | 4,542             | 2,947           | 17          | 166,400       |
|                     | 64                  | 5,137             | 3,261           | 27          | 192,000       |
|                     | 72                  | 5,267             | 3,316           | 28          | 196,096       |

#### 14.7.6. QDR II and QDR II+ Resource Utilization in Arria V Devices

The following table shows typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus Prime software for Arria V devices.

**Table 125. Resource Utilization in Arria V Devices**

| PHY Rate          | Memory Width (Bits) | Combinational ALUTs | Logic Registers | M10K Blocks | Memory (Bits) | Hard Memory Controller |
|-------------------|---------------------|---------------------|-----------------|-------------|---------------|------------------------|
| <b>Controller</b> |                     |                     |                 |             |               |                        |
| Half              | 9                   | 98                  | 120             | 0           | 0             | 0                      |
|                   | 18                  | 96                  | 156             | 0           | 0             | 0                      |
|                   | 36                  | 94                  | 224             | 0           | 0             | 0                      |
| <b>PHY</b>        |                     |                     |                 |             |               |                        |
| Half              | 9                   | 234                 | 257             | 0           | 0             | 0                      |
|                   | 18                  | 328                 | 370             | 0           | 0             | 0                      |
|                   | 36                  | 522                 | 579             | 0           | 0             | 0                      |
| <b>Total</b>      |                     |                     |                 |             |               |                        |
| Half              | 9                   | 416                 | 377             | 0           | 0             | 0                      |
|                   | 18                  | 542                 | 526             | 0           | 0             | 0                      |
|                   | 36                  | 804                 | 803             | 0           | 0             | 0                      |



#### 14.7.7. QDR II and QDR II+ Resource Utilization in Arria II GX Devices

The following table shows typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus Prime software for Arria II GX devices.

**Table 126. Resource Utilization in Arria II GX Devices**

| PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Memory (Bits) | M9K Blocks |
|----------|---------------------|---------------------|-----------------|---------------|------------|
| Half     | 9                   | 620                 | 701             | 0             | 0          |
|          | 18                  | 921                 | 1122            | 0             | 0          |
|          | 36                  | 1534                | 1964            | 0             | 0          |
| Full     | 9                   | 584                 | 708             | 0             | 0          |
|          | 18                  | 850                 | 1126            | 0             | 0          |
|          | 36                  | 1387                | 1962            | 0             | 0          |

#### 14.7.8. QDR II and QDR II+ Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices

The following table shows typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus Prime software for Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V devices.

**Table 127. Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices**

| PHY Rate | Memory Width (Bits) | Combinational ALUTS | Logic Registers | Memory (Bits) | M9K Blocks |
|----------|---------------------|---------------------|-----------------|---------------|------------|
| Half     | 9                   | 602                 | 641             | 0             | 0          |
|          | 18                  | 883                 | 1002            | 0             | 0          |
|          | 36                  | 1457                | 1724            | 0             | 0          |
| Full     | 9                   | 586                 | 708             | 0             | 0          |
|          | 18                  | 851                 | 1126            | 0             | 0          |
|          | 36                  | 1392                | 1962            | 0             | 0          |

#### 14.7.9. RLDRAM II Resource Utilization in Arria V Devices

The following table shows typical resource usage of the RLDRAM II controller with UniPHY in the current version of Quartus Prime software for Arria V devices.

**Table 128. Resource Utilization in Arria V Devices (Part 1 of 2)**

| PHY Rate          | Memory Width (Bits) | Combinational ALUTs | Logic Registers | M10K Blocks | Memory (Bits) | Hard Memory Controller |
|-------------------|---------------------|---------------------|-----------------|-------------|---------------|------------------------|
| <b>Controller</b> |                     |                     |                 |             |               |                        |
| Half              | 9                   | 353                 | 303             | 1           | 288           | 0                      |
|                   | 18                  | 350                 | 324             | 2           | 576           | 0                      |
|                   | 36                  | 350                 | 402             | 4           | 1152          | 0                      |

*continued...*



| PHY Rate     | Memory Width (Bits) | Combinational ALUTs | Logic Registers | M10K Blocks | Memory (Bits) | Hard Memory Controller |
|--------------|---------------------|---------------------|-----------------|-------------|---------------|------------------------|
| <b>PHY</b>   |                     |                     |                 |             |               |                        |
| Half         | 9                   | 295                 | 474             | 0           | 0             | 0                      |
|              | 18                  | 428                 | 719             | 0           | 0             | 0                      |
|              | 36                  | 681                 | 1229            | 0           | 0             | 0                      |
| <b>Total</b> |                     |                     |                 |             |               |                        |
| Half         | 9                   | 705                 | 777             | 1           | 288           | 0                      |
|              | 18                  | 871                 | 1043            | 2           | 576           | 0                      |
|              | 36                  | 1198                | 1631            | 4           | 1152          | 0                      |

#### 14.7.10. RLDRAM II Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices

The following table shows typical resource usage of the RLDRAM II controller with UniPHY in the current version of Quartus Prime software for Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V devices.

**Table 129. Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices <sup>(1)</sup>**

| PHY Rate | Memory Width (Bits) | Combinational ALUTs | Logic Registers | Memory (Bits) | M9K Blocks |
|----------|---------------------|---------------------|-----------------|---------------|------------|
| Half     | 9                   | 829                 | 763             | 288           | 1          |
|          | 18                  | 1145                | 1147            | 576           | 2          |
|          | 36                  | 1713                | 1861            | 1152          | 4          |
| Full     | 9                   | 892                 | 839             | 288           | 1          |
|          | 18                  | 1182                | 1197            | 576           | 1          |
|          | 36                  | 1678                | 1874            | 1152          | 2          |

Note to Table:

1. Half-rate designs use the same amount of memory as full-rate designs, but the data is organized in a different way (half the width, double the depth) and the design may need more M9K resources.

#### 14.8. Document Revision History

| Date          | Version    | Changes  |
|---------------|------------|--|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release.   |
| May 2016      | 2016.05.02 | Maintenance release.   |
| November 2015 | 2015.11.02 | Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i> . |
| May 2015      | 2015.05.04 | Maintenance release.   |
| December 2014 | 2014.12.15 | Maintenance release.   |
| August 2014   | 2014.08.15 | Removed occurrences of MegaWizard Plug-In Manager.               |

*continued...*



| Date          | Version    | Changes  |
|---------------|------------|--|
| December 2013 | 2013.12.16 | <ul style="list-style-type: none"> <li>Removed references to ALTMEMPHY.</li> <li>Removed references to HardCopy.</li> </ul>  |
| November 2012 | 2.1        | <ul style="list-style-type: none"> <li>Added RLDRAM 3 support.</li> <li>Added LRDIMM support.</li> <li>Added Arria V GZ support.</li> <li>Changed chapter number from 8 to 10.</li> </ul>  |
| June 2012     | 2.0        | <ul style="list-style-type: none"> <li>Added LPDDR2 support.</li> <li>Moved <i>Protocol Support Matrix</i> to Volume 1.</li> <li>Added Feedback icon.</li> </ul>   |
| November 2011 | 1.1        | <ul style="list-style-type: none"> <li>Combined Release Information, Device Family Support, Features list, and Unsupported Features list for DDR2, DDR3, QDR II, and RLDRAM II.</li> <li>Added Protocol Support Matrix.</li> <li>Combined Resource Utilization information for DDR2, DDR3, QDR II, and RLDRAM II.</li> <li>Updated data for 11.1.</li> </ul> |

## 15. Latency for UniPHY IP

Intel defines read and write latencies in terms of memory device clock cycles, which are always full-rate. There are two types of latencies that exists while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.

Latency of the memory interface depends on its configuration and traffic patterns, therefore you should simulate your system to determine precise latency values. The numbers presented in this chapter are typical values meant only as guidelines.

Latency found in simulation may differ from latency found on the board, because functional simulation does not consider board trace delays and differences in process, voltage, and temperature. For a given design on a given board, the latency found may differ by one clock cycle (for full-rate designs), or two clock cycles (for quarter-rate or half-rate designs) upon resetting the board. The same design can yield different latencies on different boards.

**Note:** For a half-rate controller, the local side frequency is half of the memory interface frequency. For a full-rate controller, the local side frequency is equal to the memory interface frequency.

### 15.1. DDR2 SDRAM LATENCY

The following table shows the DDR2 SDRAM latency in full-rate memory clock cycles.

**Table 130. DDR2 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) <sup>(1)</sup> <sup>(2)</sup>**

| Latency in Full-Rate Memory Clock Cycles |                              |                       |                     |                 |                        |            |                           |
|--|------------------------------|-----------------------|---------------------|-----------------|------------------------|------------|---------------------------|
| Rate                                     | Controller Address & Command | PHY Address & Command | Memory Maximum Read | PHY Read Return | Controller Read Return | Round Trip | Round Trip Without Memory |
| Half                                     | 10                           | EWL: 3                | 3–7                 | 6               | 4                      | EWL: 26–30 | EWL: 23                   |
|  |                              | OWL: 4                |                     |                 |                        | OWL: 27–31 | OWL: 24                   |
| Full                                     | 5                            | 0                     | 3–7                 | 4               | 10                     | 22–26      | 19                        |

Notes to Table:

1. EWL = Even write latency
2. OWL = Odd write latency



## 15.2. DDR3 SDRAM LATENCY

The following table shows the DDR3 SDRAM latency in full-rate memory clock cycles.

**Table 131. DDR3 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) (1) (2) (3) (4)**

| Latency in Full-Rate Memory Clock Cycles           |                              |                       |                     |                 |                        |             |                           |
|--|------------------------------|-----------------------|---------------------|-----------------|------------------------|-------------|---------------------------|
| Rate   | Controller Address & Command | PHY Address & Command | Memory Maximum Read | PHY Read Return | Controller Read Return | Round Trip  | Round Trip Without Memory |
| Quarter  | 20                           | EWER : 8              | 5-11                | EWER: 16        | 8                      | EWER: 57-63 | EWER: 52                  |
|  |                              | EWOR: 8               |                     | EWOR: 17        |                        | EWOR: 58-64 | EWOR: 53                  |
|  |                              | OWER: 11              |                     | OWER: 17        |                        | OWER: 61-67 | OWER: 56                  |
|  |                              | OWOR: 11              |                     | OWOR: 14        |                        | OWOR: 58-64 | OWOR: 53                  |
| Half   | 10                           | EWER: 3               | 5-11                | EWER: 7         | 4                      | EWER: 29-35 | EWER: 24                  |
|  |                              | EWOR: 3               |                     | EWOR: 6         |                        | EWOR: 28-34 | EWOR: 23                  |
|  |                              | OWER: 4               |                     | OWER: 6         |                        | OWER: 29-35 | OWER: 24                  |
|  |                              | OWOR: 4               |                     | OWOR: 7         |                        | OWOR: 30-36 | OWOR: 25                  |
| Full   | 5                            | 0                     | 5-11                | 4               | 10                     | 24-30       | 19                        |
| Notes to Table:                                    |                              |                       |                     |                 |                        |             |                           |
| 1. EWER = Even write latency and even read latency |                              |                       |                     |                 |                        |             |                           |
| 2. EWOR = Even write latency and odd read latency  |                              |                       |                     |                 |                        |             |                           |
| 3. OWER = Odd write latency and even read latency  |                              |                       |                     |                 |                        |             |                           |
| 4. OWOR = Odd write latency and odd read latency   |                              |                       |                     |                 |                        |             |                           |

## 15.3. LPDDR2 SDRAM LATENCY

The following table shows the LPDDR2 SDRAM latency in full-rate memory clock cycles.

**Table 132. LPDDR2 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) (1) (2) (3) (4)**

| Latency in Full-Rate Memory Clock Cycles |                              |                       |                     |                 |                        |             |                           |
|--|------------------------------|-----------------------|---------------------|-----------------|------------------------|-------------|---------------------------|
| Rate                                     | Controller Address & Command | PHY Address & Command | Memory Maximum Read | PHY Read Return | Controller Read Return | Round Trip  | Round Trip Without Memory |
| Half                                     | 10                           | EWER: 3               | 5-11                | EWER: 7         | 4                      | EWER: 29-35 | EWER: 24                  |
|  |                              | EWOR: 3               |                     | EWOR: 6         |                        | EWOR: 28-34 | EWOR: 23                  |

*continued...*

| Latency in Full-Rate Memory Clock Cycles |                              |                       |                     |                 |                        |             |                           |
|--|------------------------------|-----------------------|---------------------|-----------------|------------------------|-------------|---------------------------|
| Rate                                     | Controller Address & Command | PHY Address & Command | Memory Maximum Read | PHY Read Return | Controller Read Return | Round Trip  | Round Trip Without Memory |
|  |                              | OWER: 4               |                     | OWER: 6         |                        | OWER: 29-35 | OWER: 24                  |
|  |                              | OWOR: 4               |                     | OWOR: 7         |                        | OWOR: 30-36 | OWOR: 25                  |
| Full                                     | 5                            | 0                     | 5-11                | 4               | 10                     | 24-30       | 19                        |

Notes to Table:

1. EWER = Even write latency and even read latency
2. EWOR = Even write latency and odd read latency
3. OWER = Odd write latency and even read latency
4. OWOR = Odd write latency and odd read latency

## 15.4. QDR II and QDR II+ SRAM Latency

The following table shows the latency in full-rate memory clock cycles.

**Table 133. QDR II Latency (In Full-Rate Memory Clock Cycles) <sup>(1)</sup>**

| Latency in Full-Rate Memory Clock Cycles |                              |                       |                     |                 |                        |            |                           |
|--|------------------------------|-----------------------|---------------------|-----------------|------------------------|------------|---------------------------|
| Rate                                     | Controller Address & Command | PHY Address & Command | Memory Maximum Read | PHY Read Return | Controller Read Return | Round Trip | Round Trip Without Memory |
| Half 1.5 RL                              | 2                            | 5.5                   | 1.5                 | 7.0             | 0                      | 16         | 14.5                      |
| Half 2.0 RL                              | 2                            | 5.5                   | 2.0                 | 6.5             | 0                      | 16         | 14.0                      |
| Half 2.5 RL                              | 2                            | 5.5                   | 2.5                 | 6.0             | 0                      | 16         | 13.5                      |
| Full 1.5 RL                              | 2                            | 1.5                   | 1.5                 | 4.0             | 1                      | 10         | 8.5                       |
| Full 2.0 RL                              | 2                            | 1.5                   | 2.0                 | 4.5             | 1                      | 11         | 9.0                       |
| Full 2.5 RL                              | 2                            | 1.5                   | 2.5                 | 4.0             | 1                      | 11         | 8.5                       |

Note to Table:

1. RL = Read latency

## 15.5. RLDRAM II Latency

The following table shows the latency in full-rate memory clock cycles.



**Table 134. RLDRAM II Latency (In Full-Rate Memory Clock Cycles) (1) (2)**

| Latency in Full-Rate Memory Clock Cycles |                              |                       |                     |                 |                        |            |                           |
|--|------------------------------|-----------------------|---------------------|-----------------|------------------------|------------|---------------------------|
| Rate                                     | Controller Address & Command | PHY Address & Command | Memory Maximum Read | PHY Read Return | Controller Read Return | Round Trip | Round Trip Without Memory |
| Half                                     | 4                            | EWL: 1                | 3-8                 | EWL: 4          | 0                      | EWL: 12-17 | EWL: 9                    |
|  |                              | OWL: 2                |                     | OWL: 4          |                        | OWL: 13-18 | OWL: 10                   |
| Full                                     | 2                            | 1                     | 3-8                 | 4               | 0                      | 10-15      | 7                         |

Notes to Table:

1. EWL = Even write latency
2. OWL = Odd write latency

## 15.6. RLDRAM 3 Latency

The following table shows the latency in full-rate memory clock cycles.

**Table 135. RLDRAM 3 Latency (In Full-Rate Memory Clock Cycles)**

| Latency in Full-Rate Memory Clock Cycles |                       |                     |                 |                        |            |                           |
|--|-----------------------|---------------------|-----------------|------------------------|------------|---------------------------|
| Rate                                     | PHY Address & Command | Memory Maximum Read | PHY Read Return | Controller Read Return | Round Trip | Round Trip Without Memory |
| Quarter                                  | 7                     | 3-16                | 18              | 0                      | 28-41      | 25                        |
| Half                                     | 4                     | 3-16                | 6               | 0                      | 13-26      | 10                        |

## 15.7. Variable Controller Latency

The variable controller latency feature allows you to take advantage of lower latency for variations designed to run at lower frequency. When deciding whether to vary the controller latency from the default value of 1, be aware of the following considerations:

- Reduced latency can help achieve a reduction in resource usage and clock cycles in the controller, but might result in lower fMAX.
- Increased latency can help achieve greater fMAX, but might consume more clock cycles in the controller and result in increased resource usage.

If you select a latency value that is inappropriate for the target frequency, the system displays a warning message in the text area at the bottom of the parameter editor.

You can change the controller latency by altering the value of the **Controller Latency** setting in the **Controller Settings** section of the **General Settings** tab of the QDR II and QDR II+ SRAM controller with UniPHY parameter editor.

## 15.8. Document Revision History



| Date          | Version    | Changes   |
|---------------|------------|---|
| May 2017      | 2017.05.08 | Rebranded as Intel.   |
| October 2016  | 2016.10.31 | Maintenance release.  |
| May 2016      | 2016.05.02 | Maintenance release.  |
| November 2015 | 2015.11.02 | Maintenance release.  |
| May 2015      | 2015.05.04 | Maintenance release.  |
| December 2014 | 2014.12.15 | Maintenance release.  |
| August 2014   | 2014.08.15 | Maintenance release.  |
| December 2013 | 2013.12.16 | Updated latency data for QDR II/II+.  |
| November 2012 | 2.1        | <ul style="list-style-type: none"><li>Added latency information for RLDRAM 3.</li><li>Changed chapter number from 9 to 11.</li></ul>  |
| June 2012     | 2.0        | <ul style="list-style-type: none"><li>Added latency information for LPDDR2.</li><li>Added Feedback icon.</li></ul>  |
| November 2011 | 1.0        | <ul style="list-style-type: none"><li>Consolidated latency information from 11.0 DDR2 and DDR3 SDRAM Controller with UniPHY User Guide, QDR II and QDR II+ SRAM Controller with UniPHY User Guide, and RLDRAM II Controller with UniPHY IP User Guide.</li><li>Updated data for 11.1.</li></ul> |

## 16. Timing Diagrams for UniPHY IP

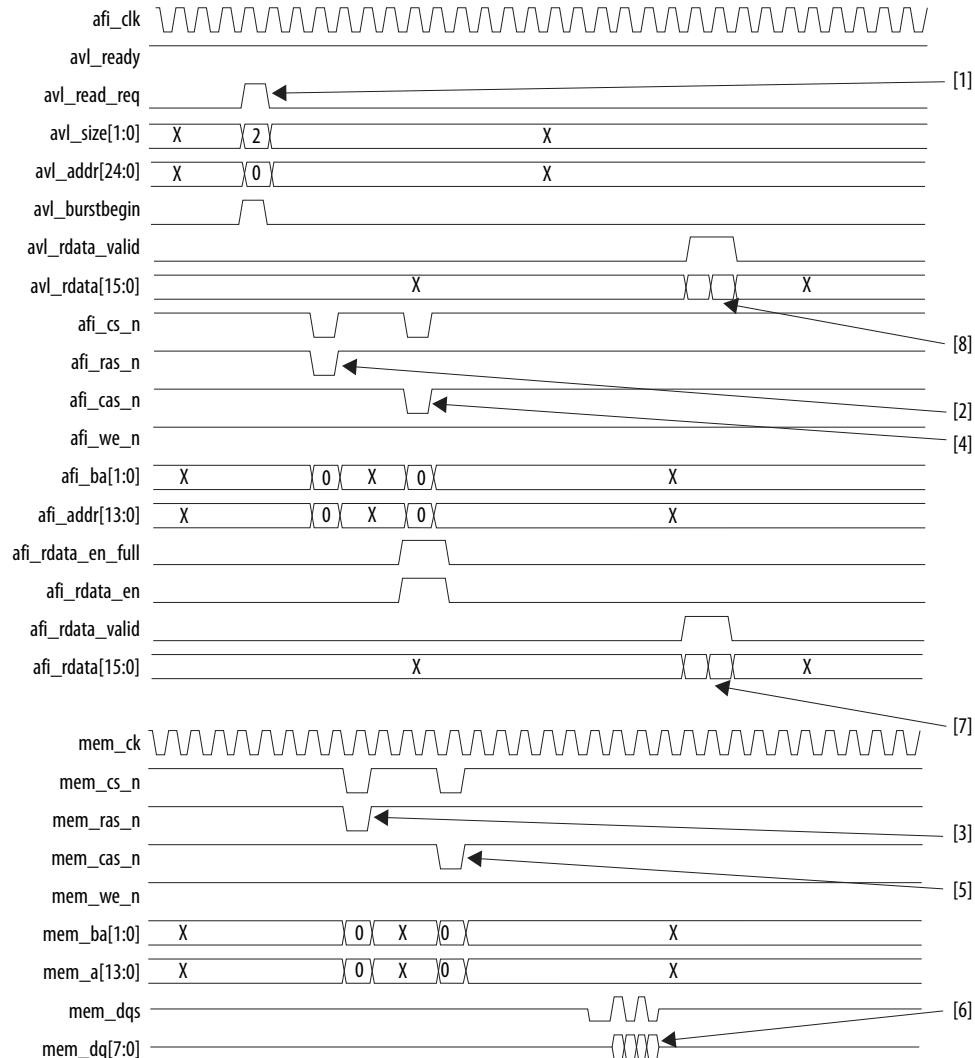
The following topics contain timing diagrams for UniPHY-based external memory interface IP for supported protocols.

### 16.1. DDR2 Timing Diagrams

This topic contains timing diagrams for UniPHY-based external memory interface IP for DDR2 protocols.

The following figures present timing diagrams based on a Stratix III device:

**Figure 183. Full-Rate DDR2 SDRAM Read**

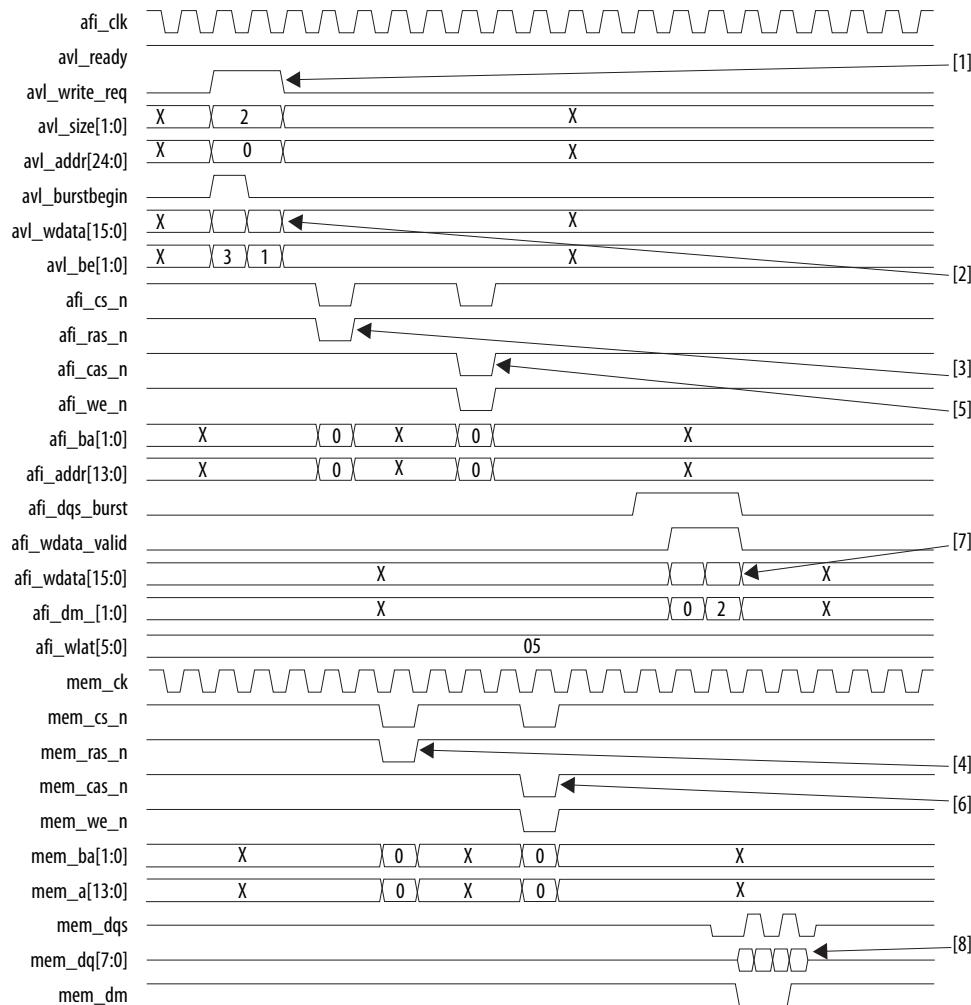


Notes for the above Figure:

1. Controller receives read command.
2. Controller issues activate command to PHY.
3. PHY issues activate command to memory.
4. Controller issues read command to PHY.
5. PHY issues read command to memory.
6. PHY receives read data from memory.
7. Controller receives read data from PHY.
8. User logic receives read data from controller.

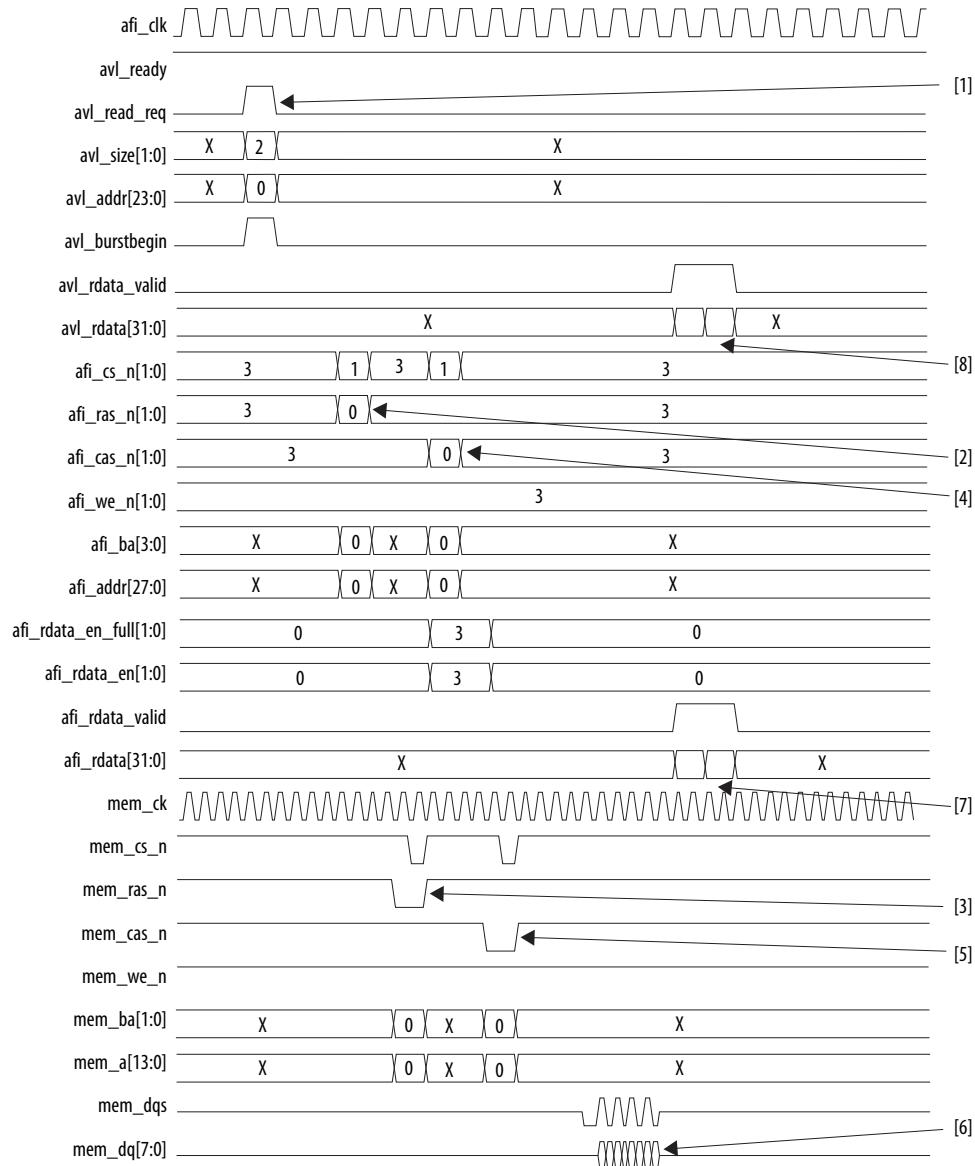


**Figure 184. Full-Rate DDR2 SDRAM Write**



Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues activate command to PHY.
4. PHY issues activate command to memory.
5. Controller issues write command to PHY.
6. PHY issues write command to memory.
7. Controller sends write data to PHY.
8. PHY sends write data to memory.

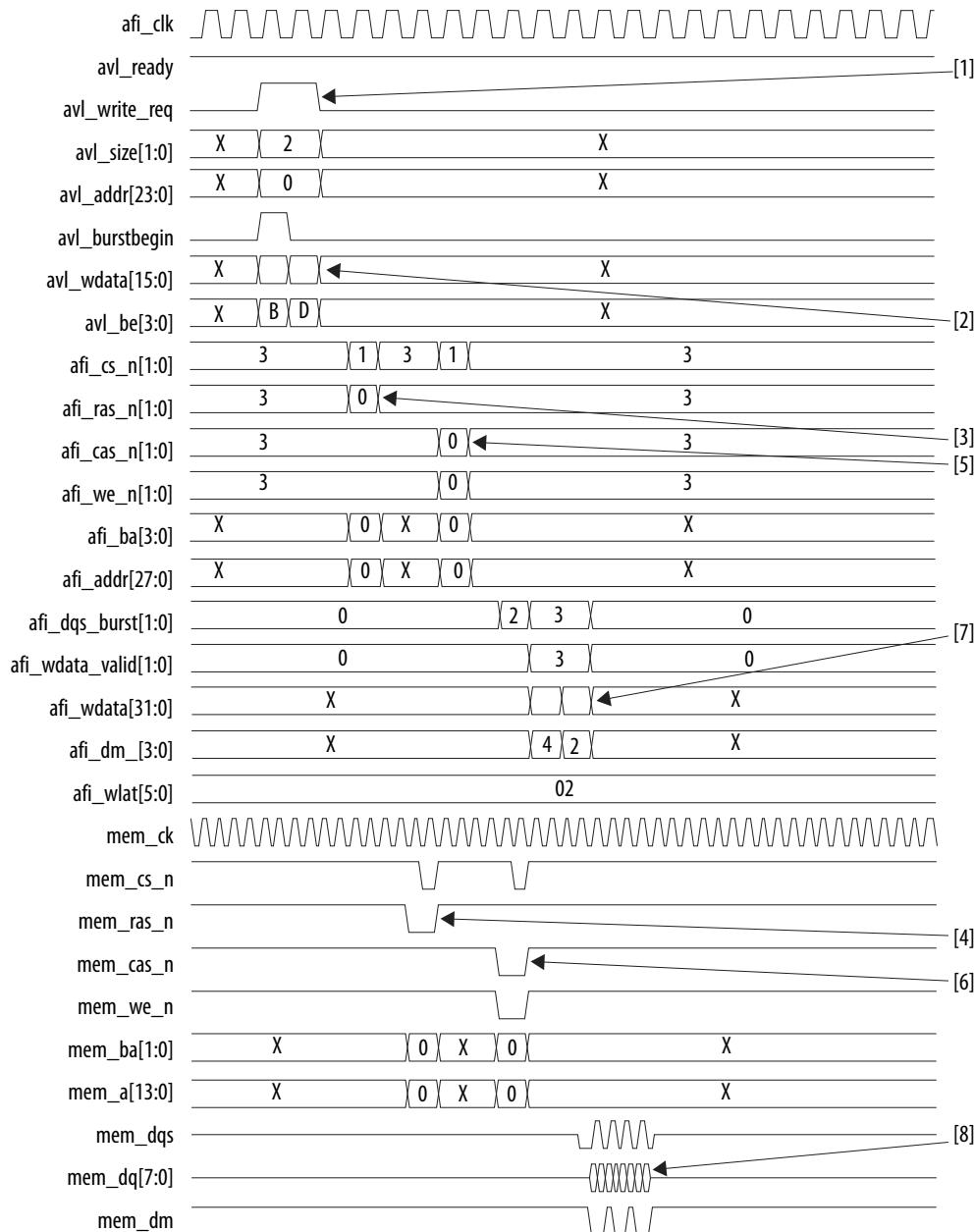
**Figure 185. Half-Rate DDR2 SDRAM Read**


Notes for the above Figure:

1. Controller receives read command.
2. Controller issues activate command to PHY.
3. PHY issues activate command to memory.
4. Controller issues read command to PHY.
5. PHY issues read command to memory.
6. PHY receives read data from memory.
7. Controller receives read data from PHY.
8. User logic receives read data from controller.



Figure 186. Half-Rate DDR2 SDRAM Write



Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues activate command to PHY.
4. PHY issues activate command to memory.
5. Controller issues write command to PHY.



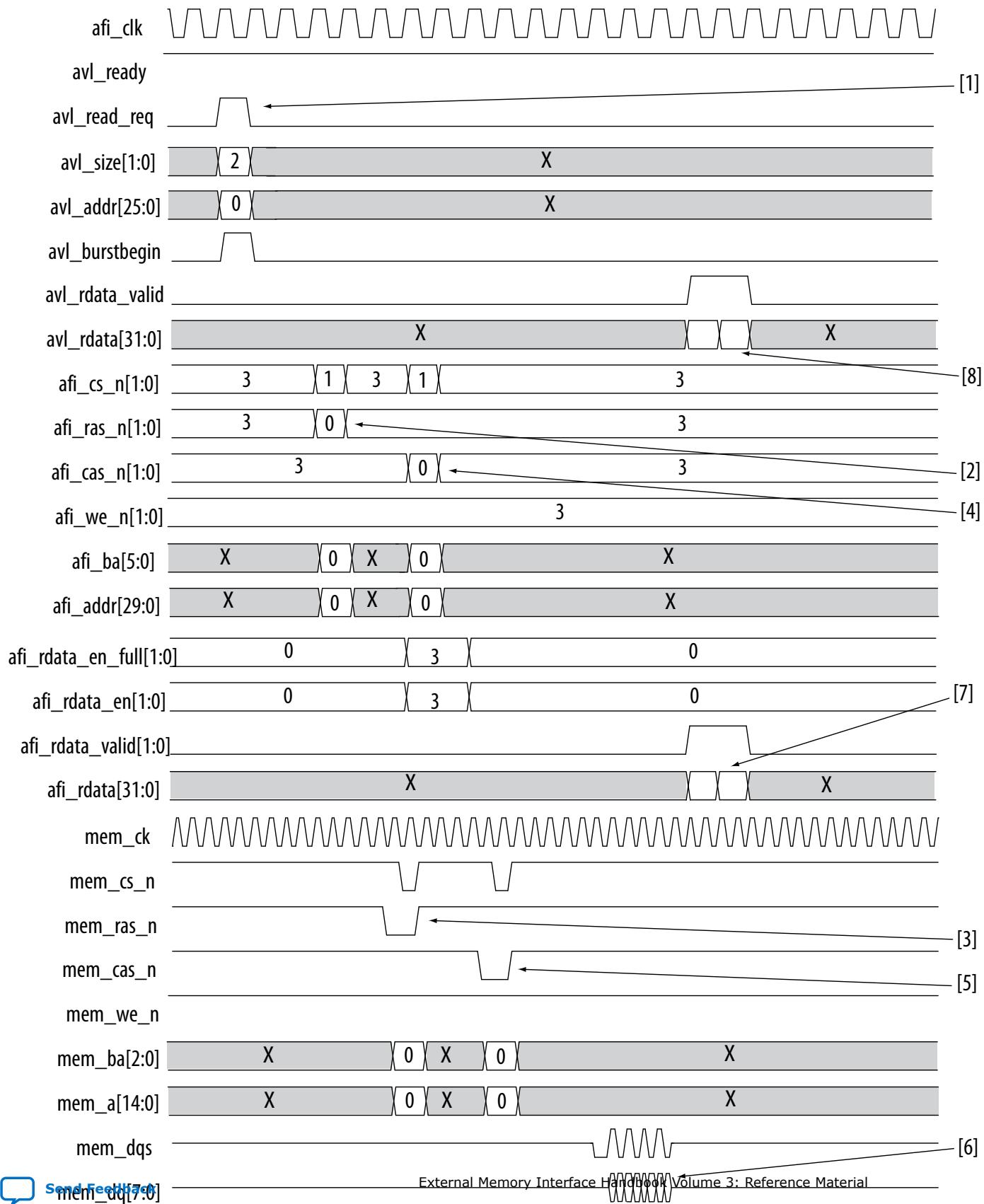
6. PHY issues write command to memory.
7. Controller sends write data to PHY.
8. PHY sends write data to memory.

## 16.2. DDR3 Timing Diagrams

This topic contains timing diagrams for UniPHY-based external memory interface IP for DDR3 protocols.

The following figures present timing diagrams based on a Stratix III device

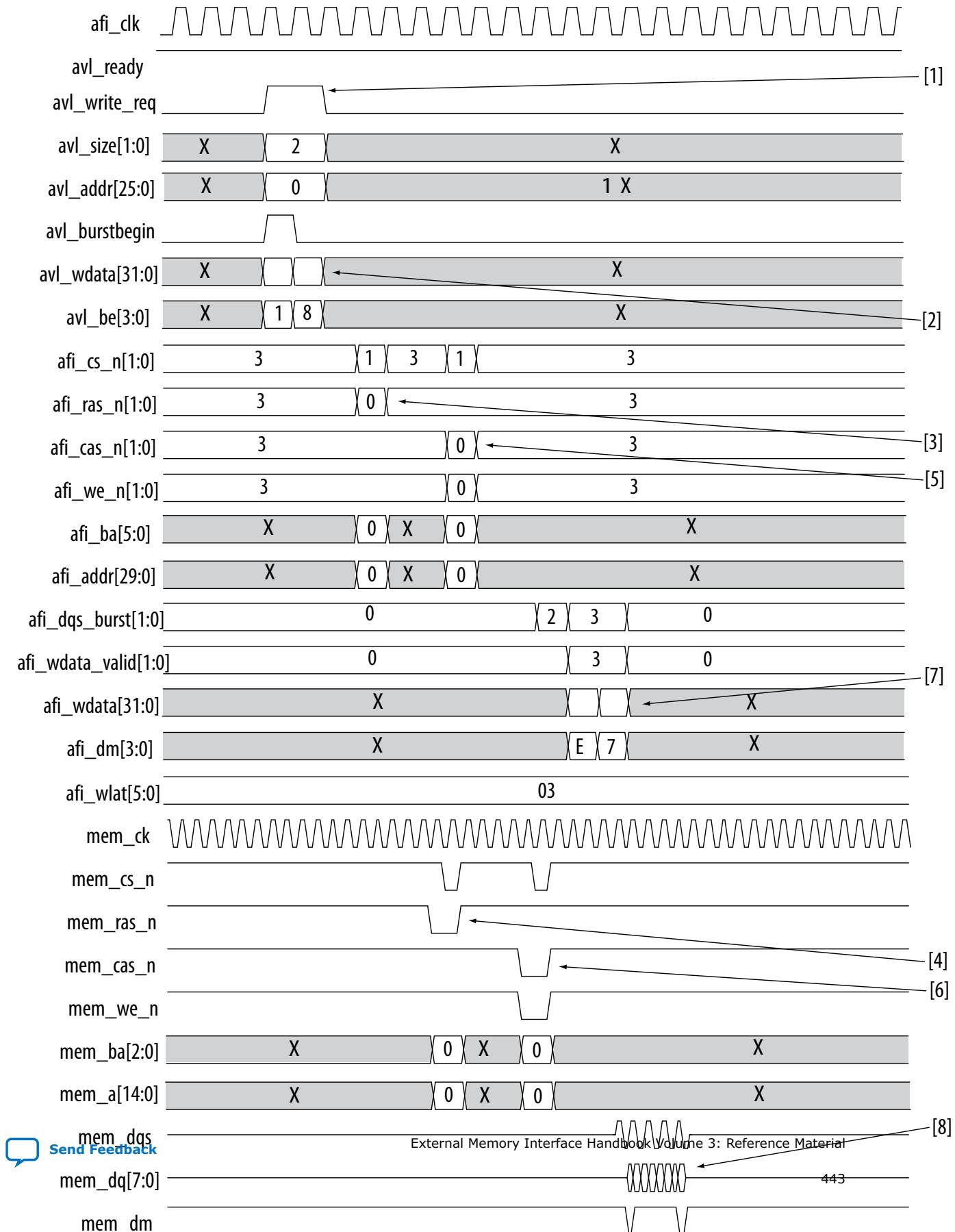
**Figure 187. Half-Rate DDR3 SDRAM Read**



Notes for the above Figure:

1. Controller receives read command.
2. Controller issues activate command to PHY.
3. PHY issues activate command to memory.
4. Controller issues read command to PHY.
5. PHY issues read command to memory.
6. PHY receives read data from memory.
7. Controller receives read data from PHY.
8. User logic receives read data from controller.

**Figure 188. Half-Rate DDR3 SDRAM Writes**

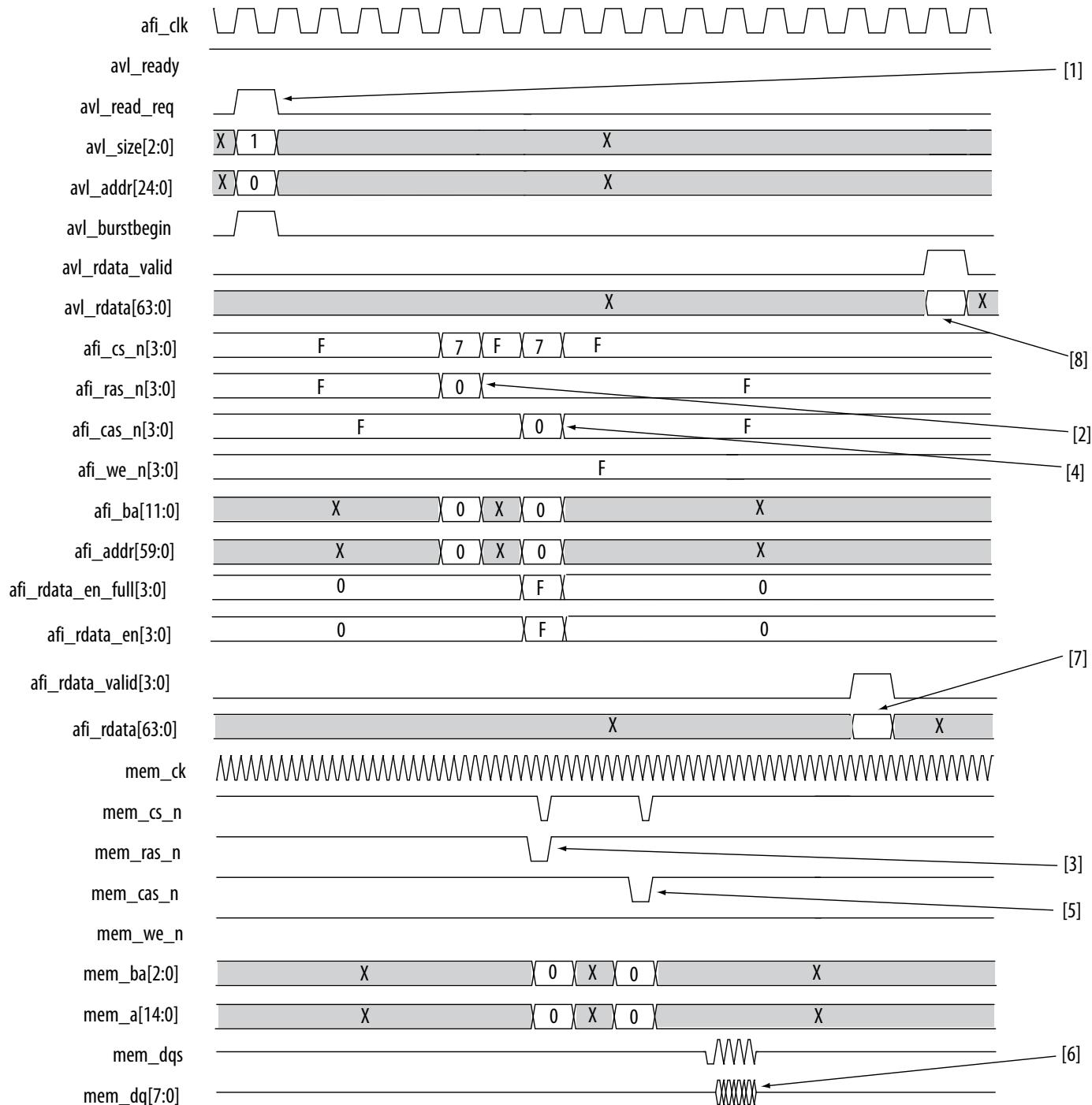


Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues activate command to PHY.
4. PHY issues activate command to memory.
5. Controller issues write command to PHY.
6. PHY issues write command to memory.
7. Controller sends write data to PHY.
8. PHY sends write data to memory.



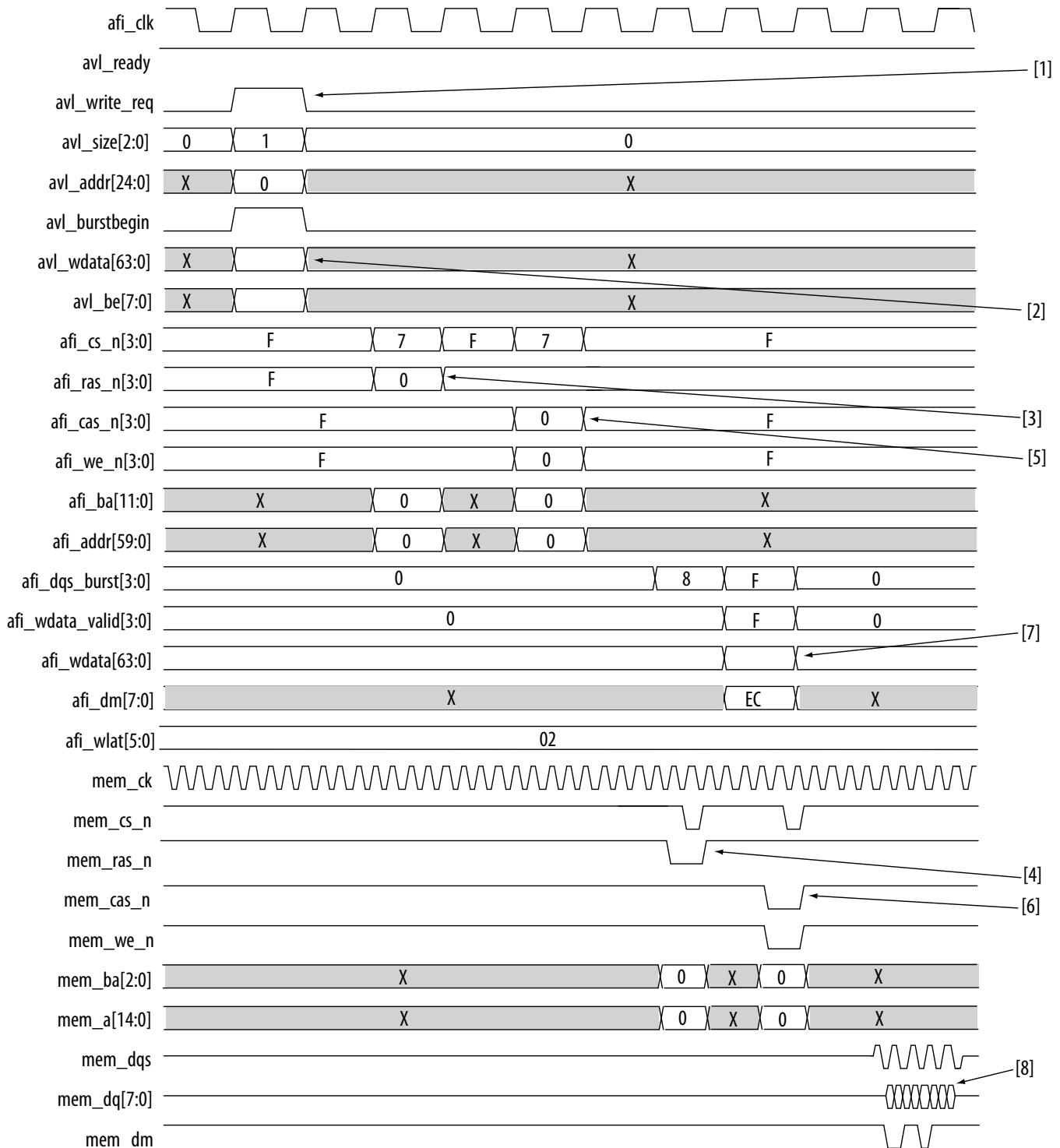
**Figure 189. Quarter-Rate DDR3 SDRAM Reads**



Notes for the above Figure:

1. Controller receives read command.
2. Controller issues activate command to PHY.
3. PHY issues activate command to memory.
4. Controller issues read command to PHY.
5. PHY issues read command to memory.
6. PHY receives read data from memory
7. Controller receives read data from PHY
8. User logic receives read data from controller.

**Figure 190. Quarter-Rate DDR3 SDRAM Writes**





Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues activate command to PHY
4. PHY issues activate command to memory.
5. Controller issues write command to PHY
6. PHY issues write command to memory
7. Controller sends write data to PHY
8. PHY sends write data to memory.

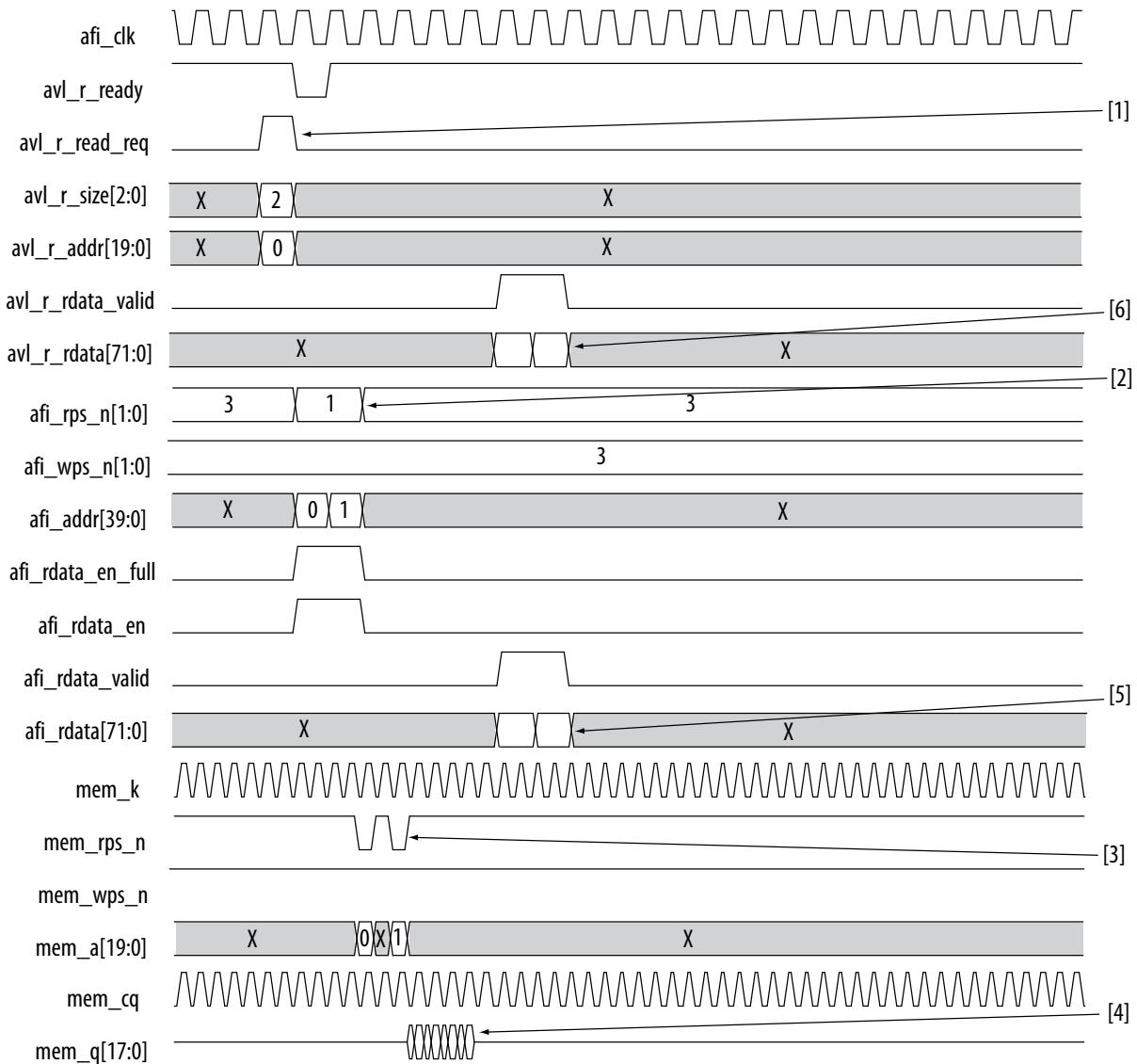
### 16.3. QDR II and QDR II+ Timing Diagrams

This topic contains timing diagrams for UniPHY-based external memory interface IP for QDR II and QDR II+ protocols.



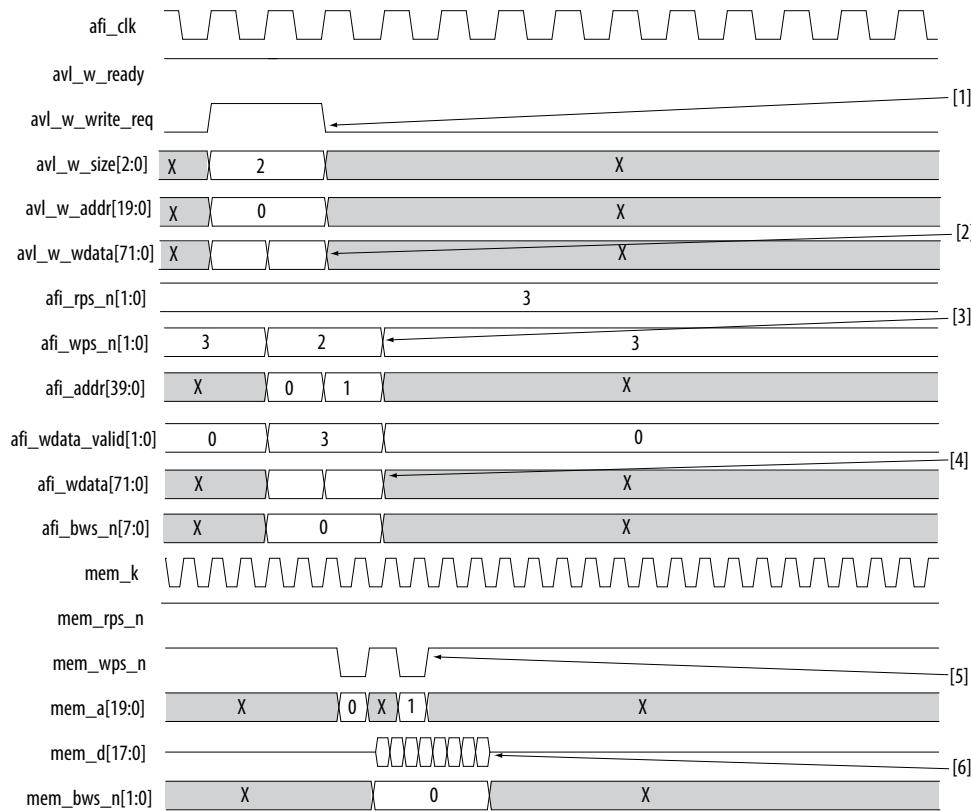
The following figures present timing diagrams, based on a Stratix III device:

**Figure 191. Half-Rate QDR II and QDR II+ SRAM Read**



Notes for the above Figure:

1. Controller receives read command.
2. Controller issues two read commands to PHY.
3. PHY issues two read commands to memory.
4. PHY receives read data from memory.
5. Controller receives read data from PHY.
6. User logic receives read data from controller.

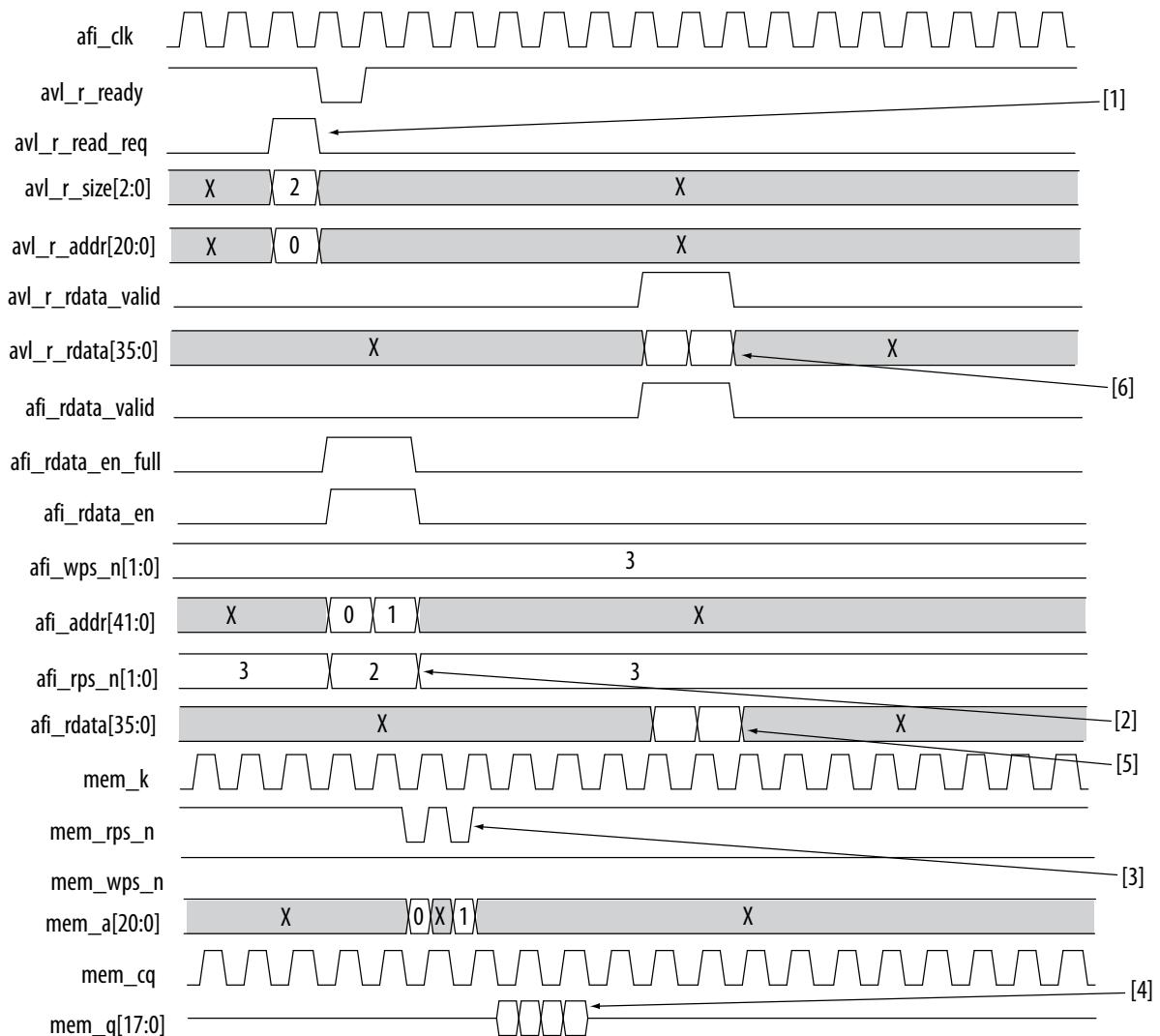
**Figure 192. Half-Rate QDR II and QDR II+ SRAM Write**


Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues two write commands to PHY.
4. Controller sends write data to PHY.
5. PHY issues two write commands to memory.
6. PHY sends write data to memory.

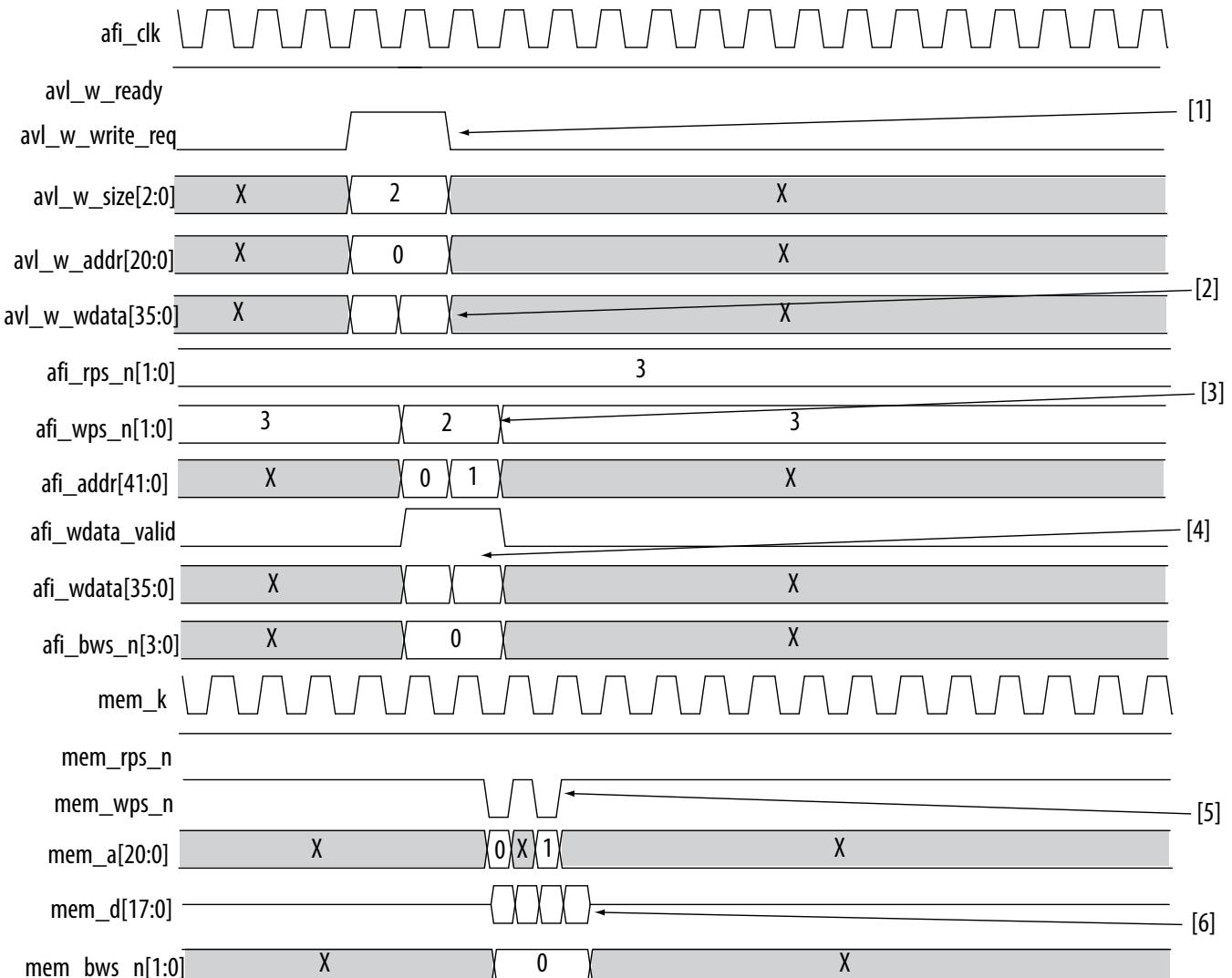


Figure 193. Full-Rate QDR II and QDR II+ SRAM Read



Notes for the above Figure:

1. Controller receives read command.
2. Controller issues two read commands to PHY.
3. PHY issues two read commands to memory.
4. PHY receives read data from memory.
5. Controller receives read data from PHY.
6. User logic receives read data from controller.

**Figure 194. Full-Rate QDR II and QDR II+ SRAM Write**


Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues two write commands to PHY.
4. Controller sends write data to PHY.
5. PHY issues two write commands to memory.
6. PHY sends write data to memory.

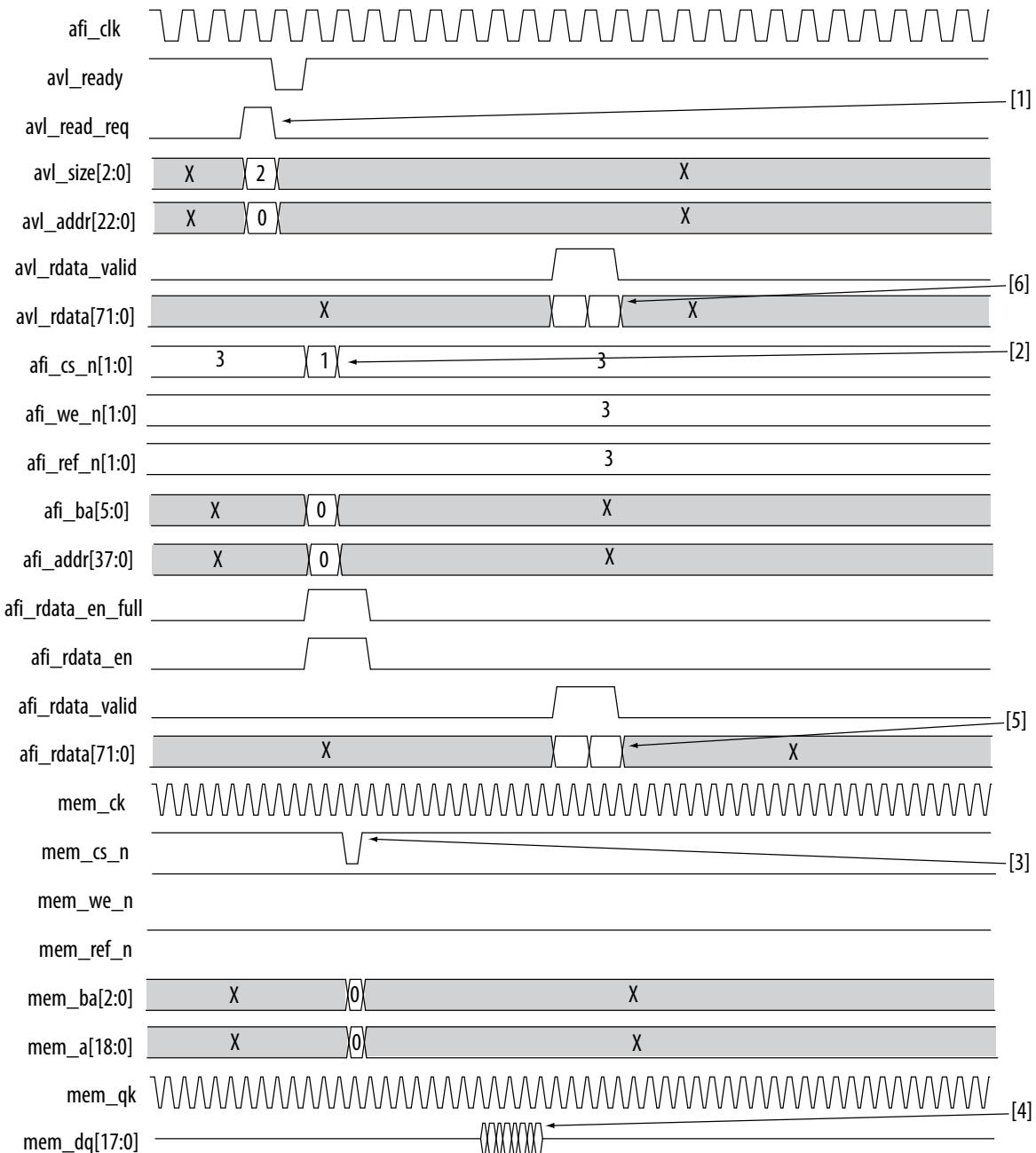
## 16.4. RLDRAM II Timing Diagrams

This topic contains timing diagrams for UniPHY-based external memory interface IP for RLDRAM protocols.



The following figures present timing diagrams, based on a Stratix III device:

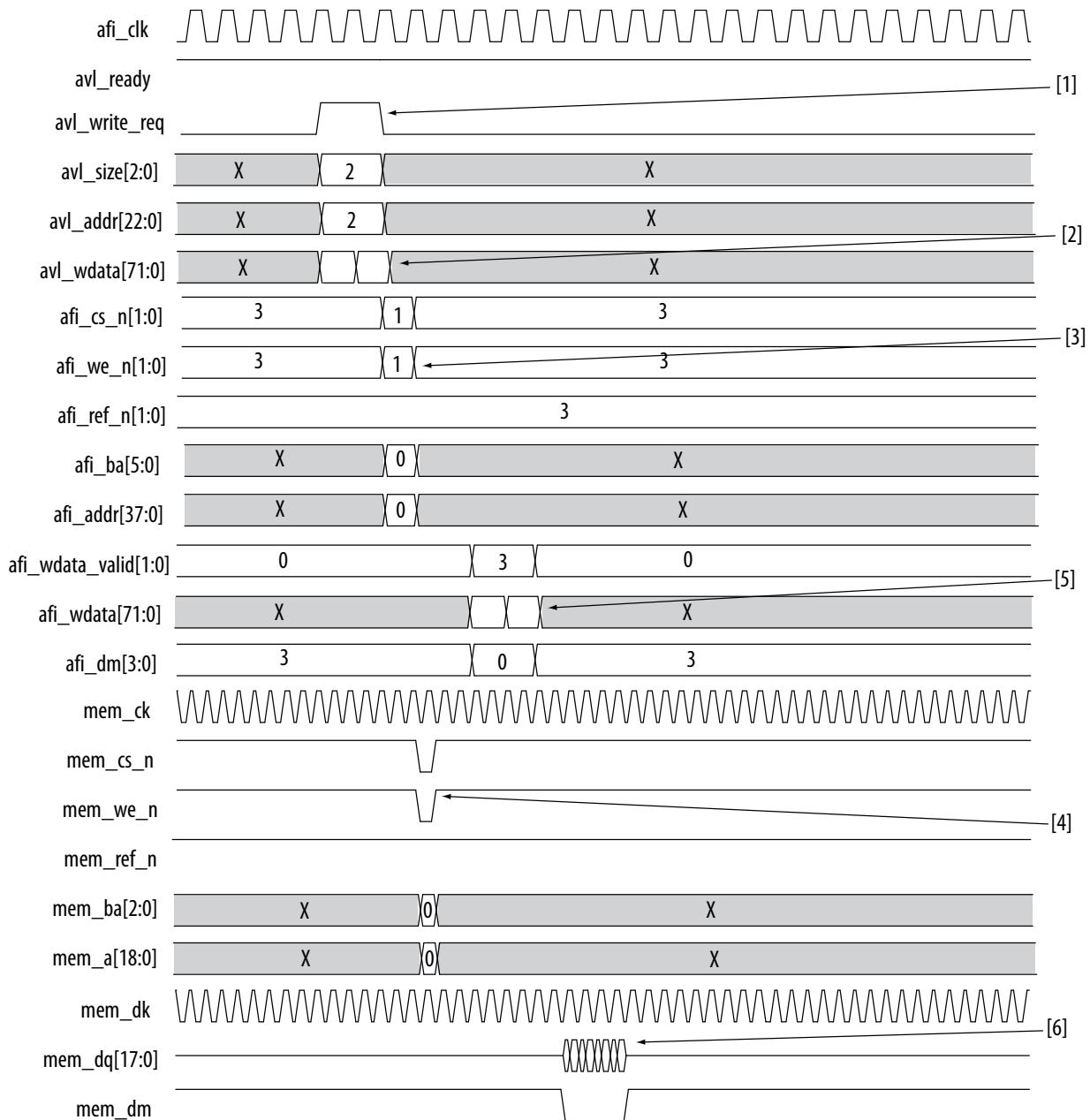
**Figure 195. Half-Rate RLDRAM II Read**



Notes for the above Figure:

1. Controller receives read command.
2. Controller issues read command to PHY.
3. PHY issues read command to memory.

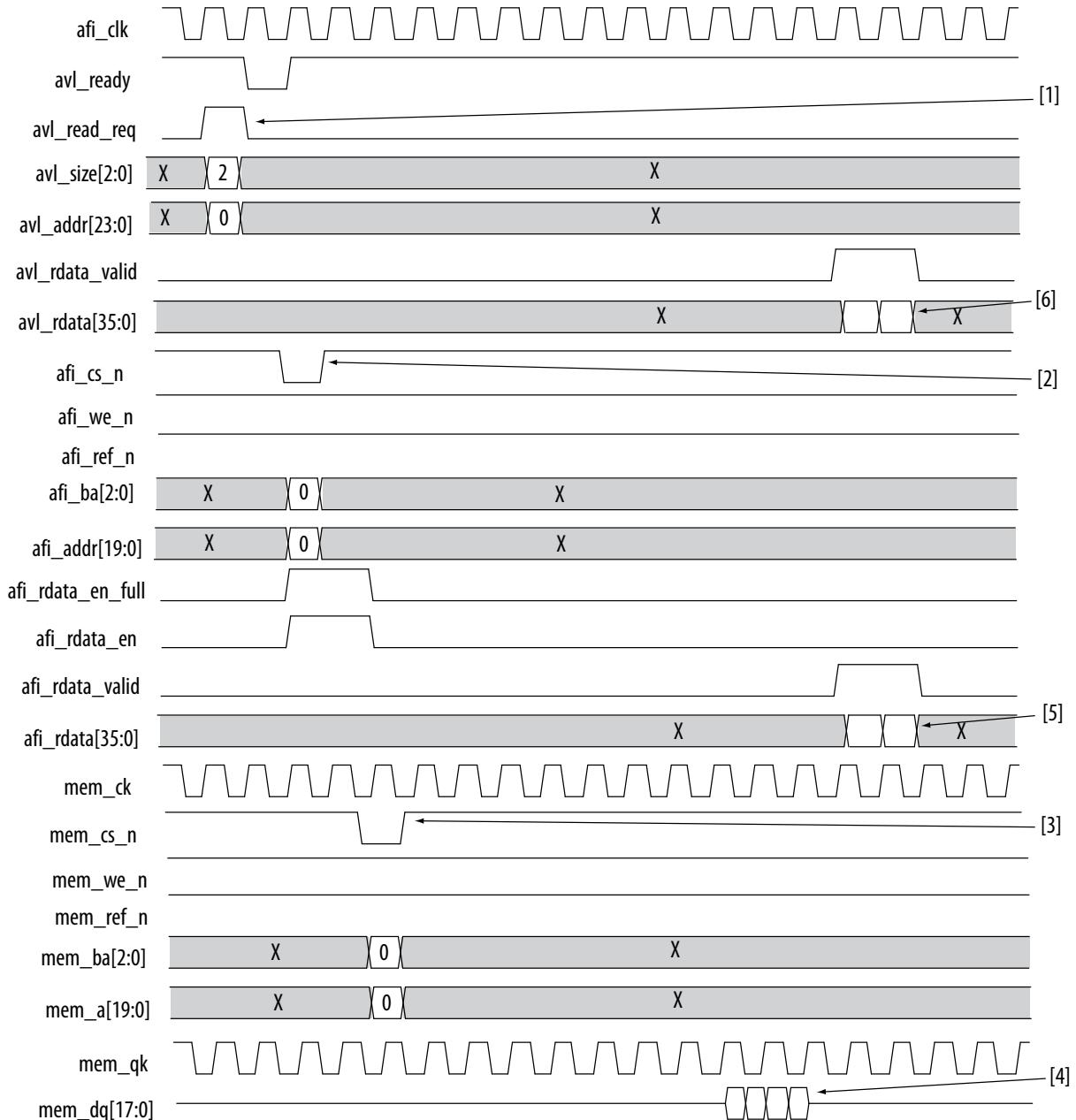
4. PHY receives read data from memory.
5. Controller receives read data from PHY.
6. User logic receives read data from controller.

**Figure 196. Half-Rate RLDRAM II Write**




Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues write command to PHY.
4. PHY issues write command to memory.
5. Controller sends write data to PHY.
6. PHY sends write data to memory.

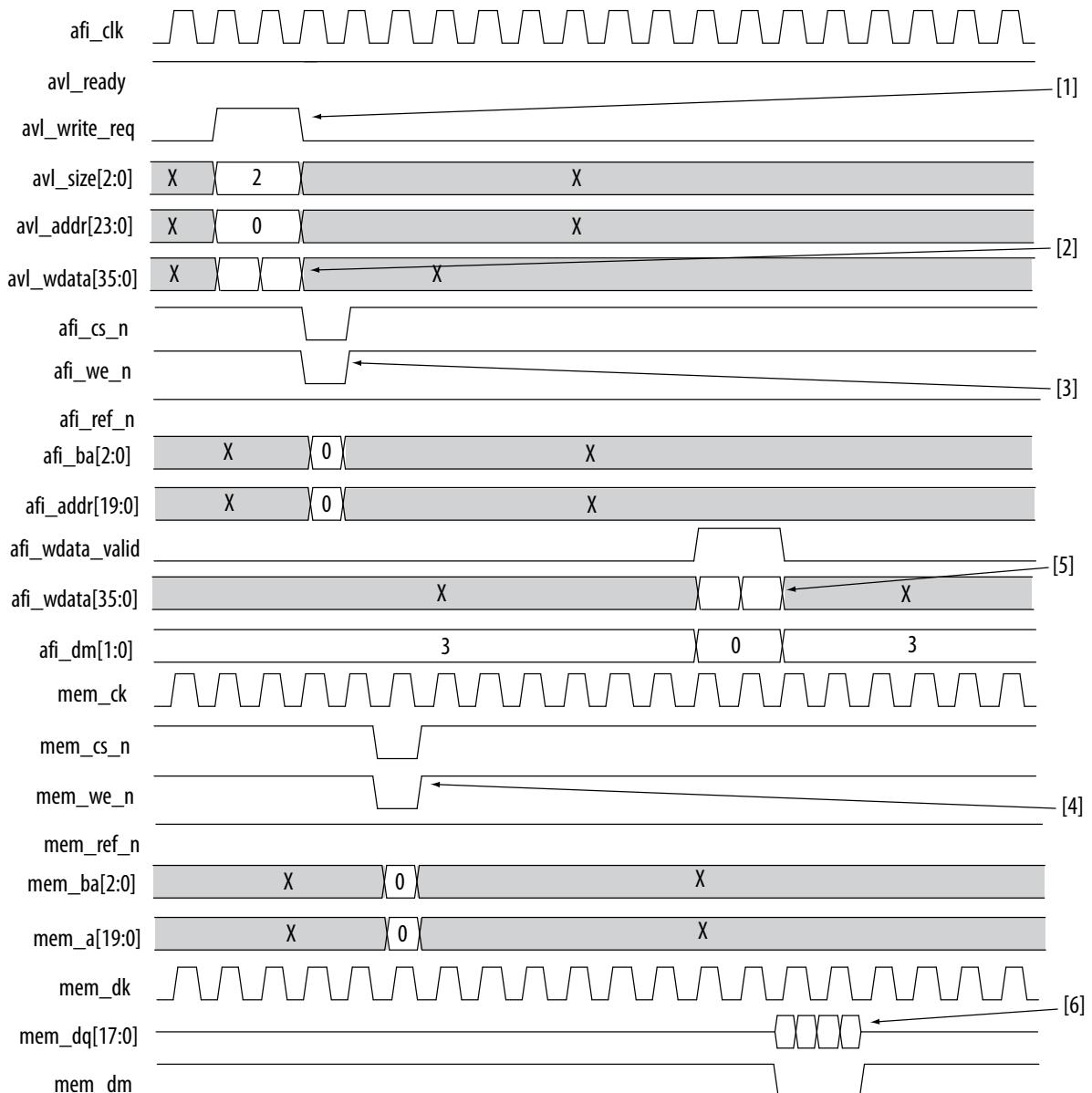
**Figure 197. Full-Rate RLDRAM II Read**


Notes for the above Figure:

1. Controller receives read command.
2. Controller issues read command to PHY.
3. PHY issues read command to memory.



4. PHY receives read data from memory.
5. Controller receives read data from PHY.
6. User logic receives read data from controller.

**Figure 198. Full-Rate RLDRAM II Write**

Notes for the above Figure:

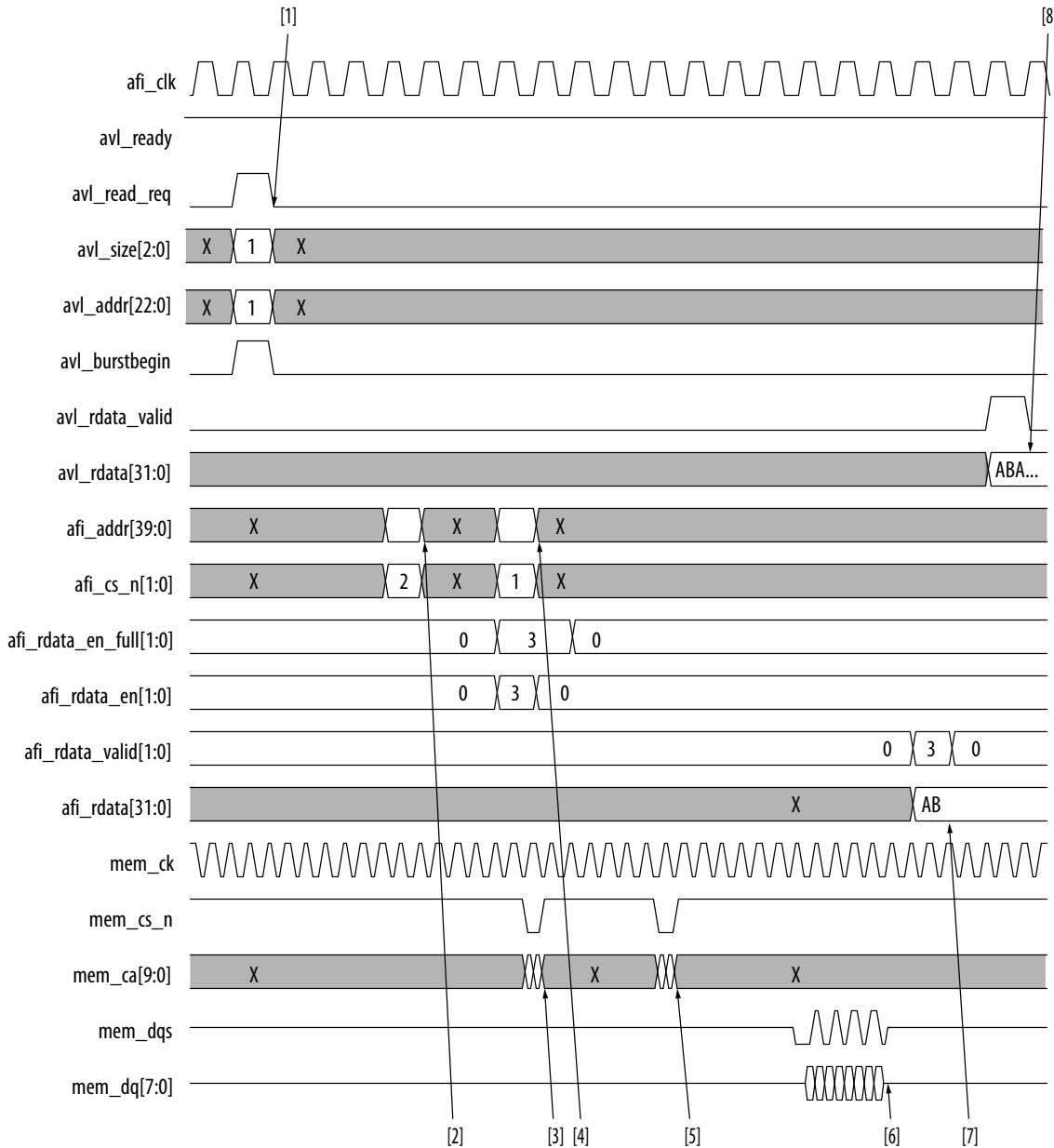
1. Controller receives write command.
2. Controller receives write data.
3. Controller issues write command to PHY.

4. PHY issues write command to memory.
5. Controller sends write data to PHY.
6. PHY sends write data to memory.

## 16.5. LPDDR2 Timing Diagrams

This topic contains timing diagrams for UniPHY-based external memory interface IP for LPDDR2 protocols.

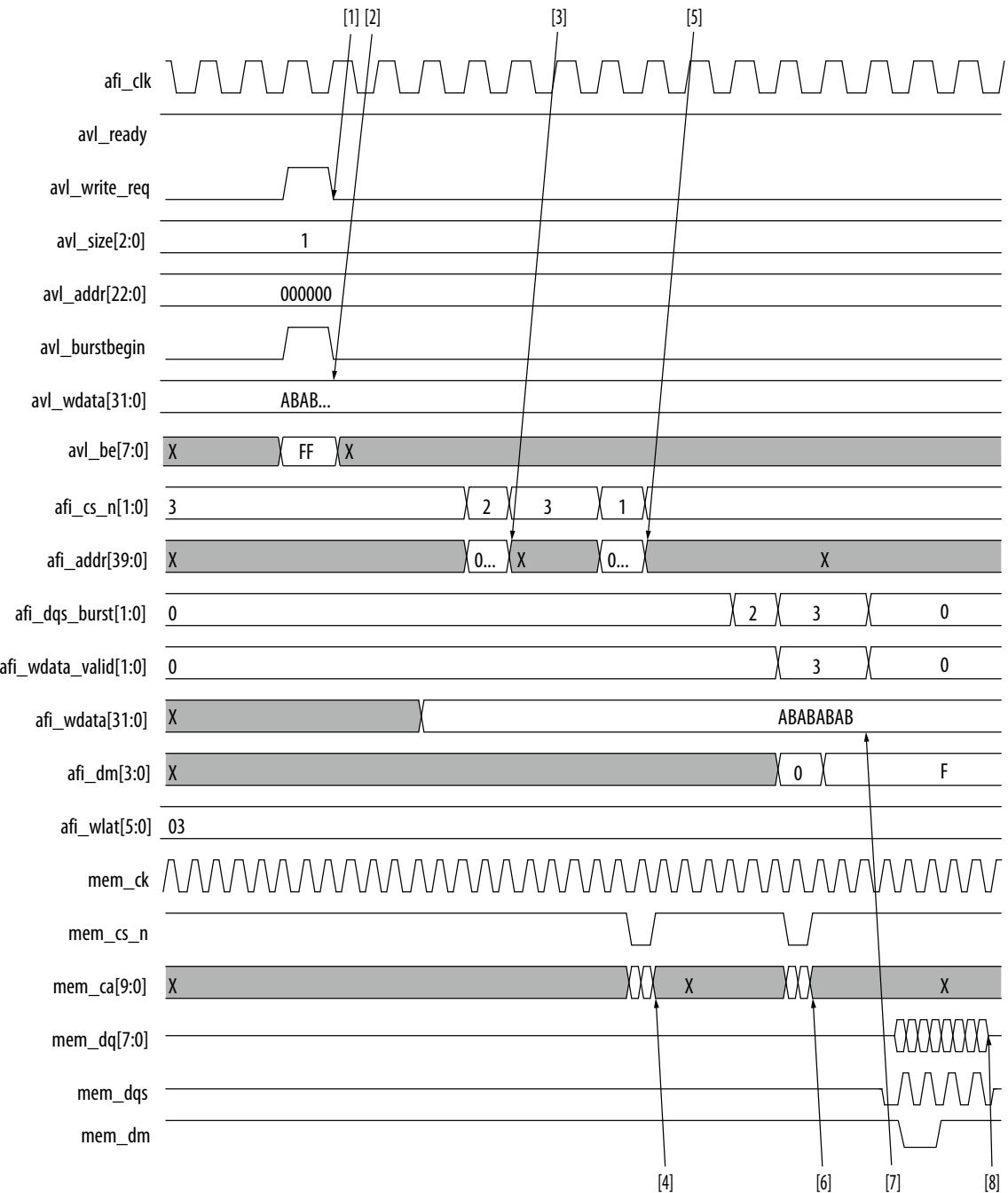
**Figure 199. Half-Rate LPDDR2 Read**





Notes for the above Figure:

1. Controller receives read command.
2. Controller issues activate command to PHY.
3. PHY issues activate command to memory.
4. Controller issues read command to PHY.
5. PHY issues read command to memory.
6. PHY receives read data from memory.
7. Controller receives read data from PHY.
8. User logic receives read data from controller.

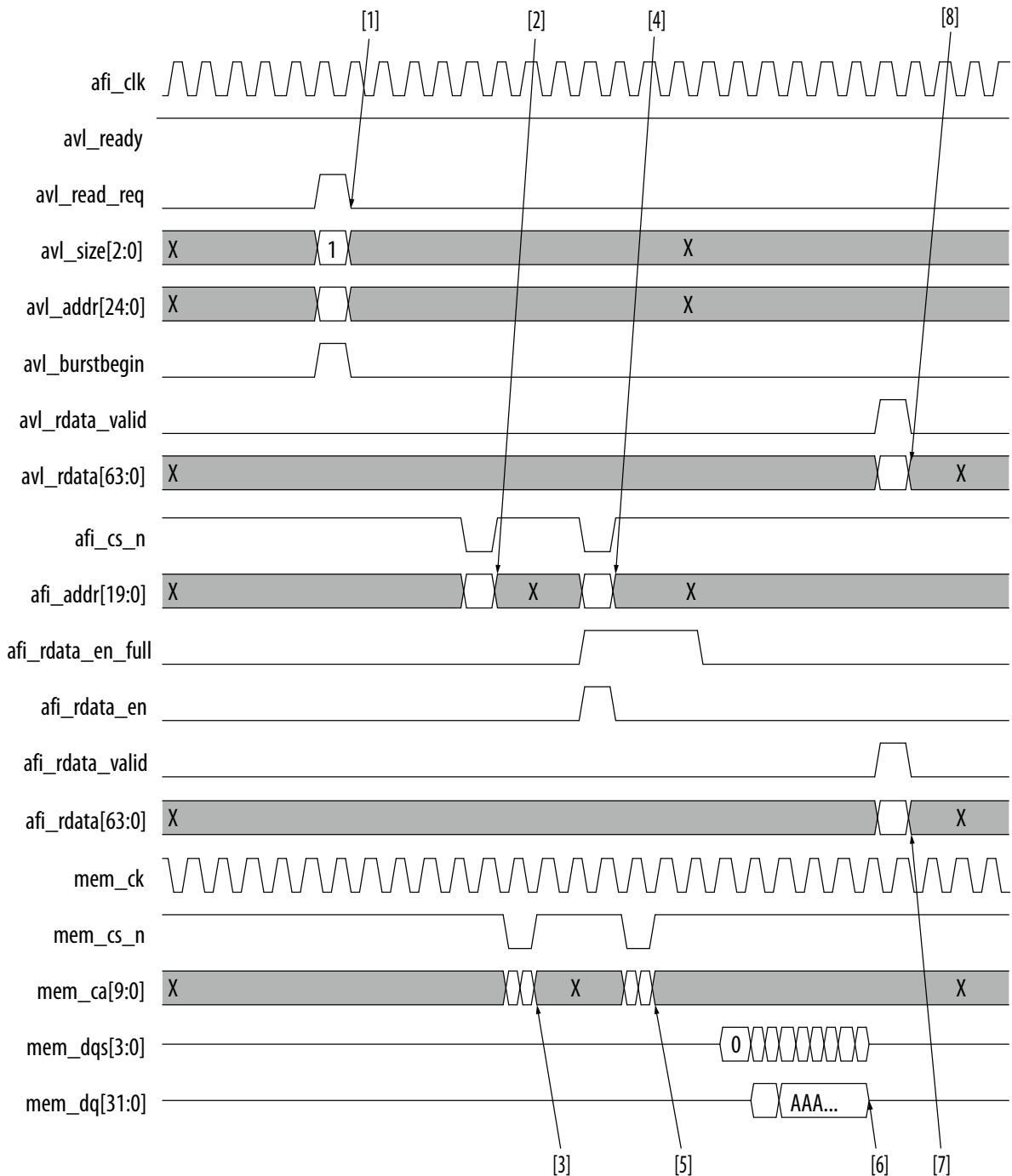
**Figure 200. Half-Rate LPDDR2 Write**


Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues activate command to PHY.



4. PHY issues activate command to memory.
5. Controller issues write command to PHY.
6. PHY issues write command to memory.
7. Controller sends write data to PHY.
8. PHY sends write data to memory.

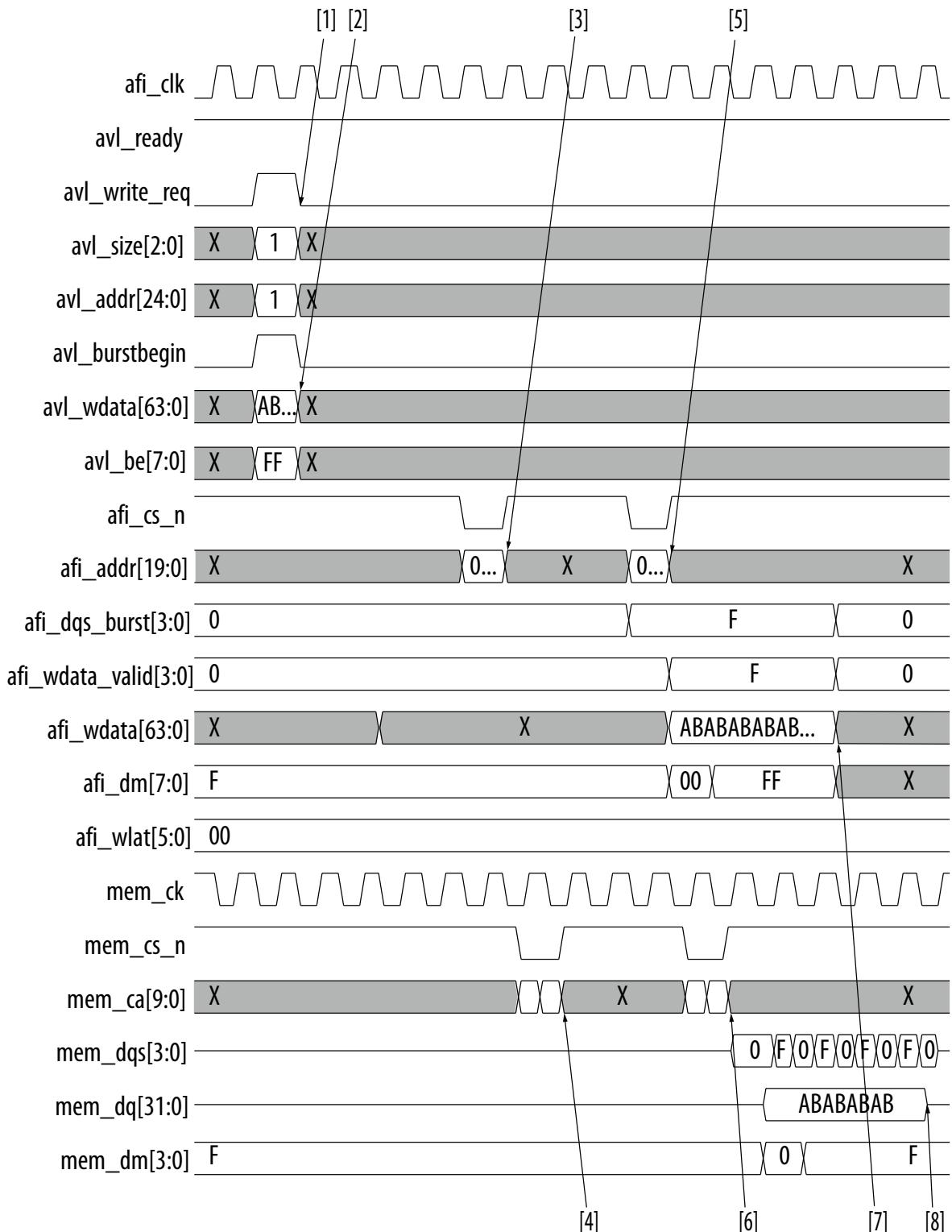
**Figure 201. Full-Rate LPDDR2 Read**


Notes for the above Figure:

1. Controller receives read command.
2. Controller issues activate command to PHY.
3. PHY issues activate command to memory.



4. Controller issues read command to PHY.
5. PHY issues read command to memory.
6. PHY receives read data from memory.
7. Controller receives read data from PHY.
8. User logic receives read data from controller.

**Figure 202. Full-Rate LPDDR2 Write**




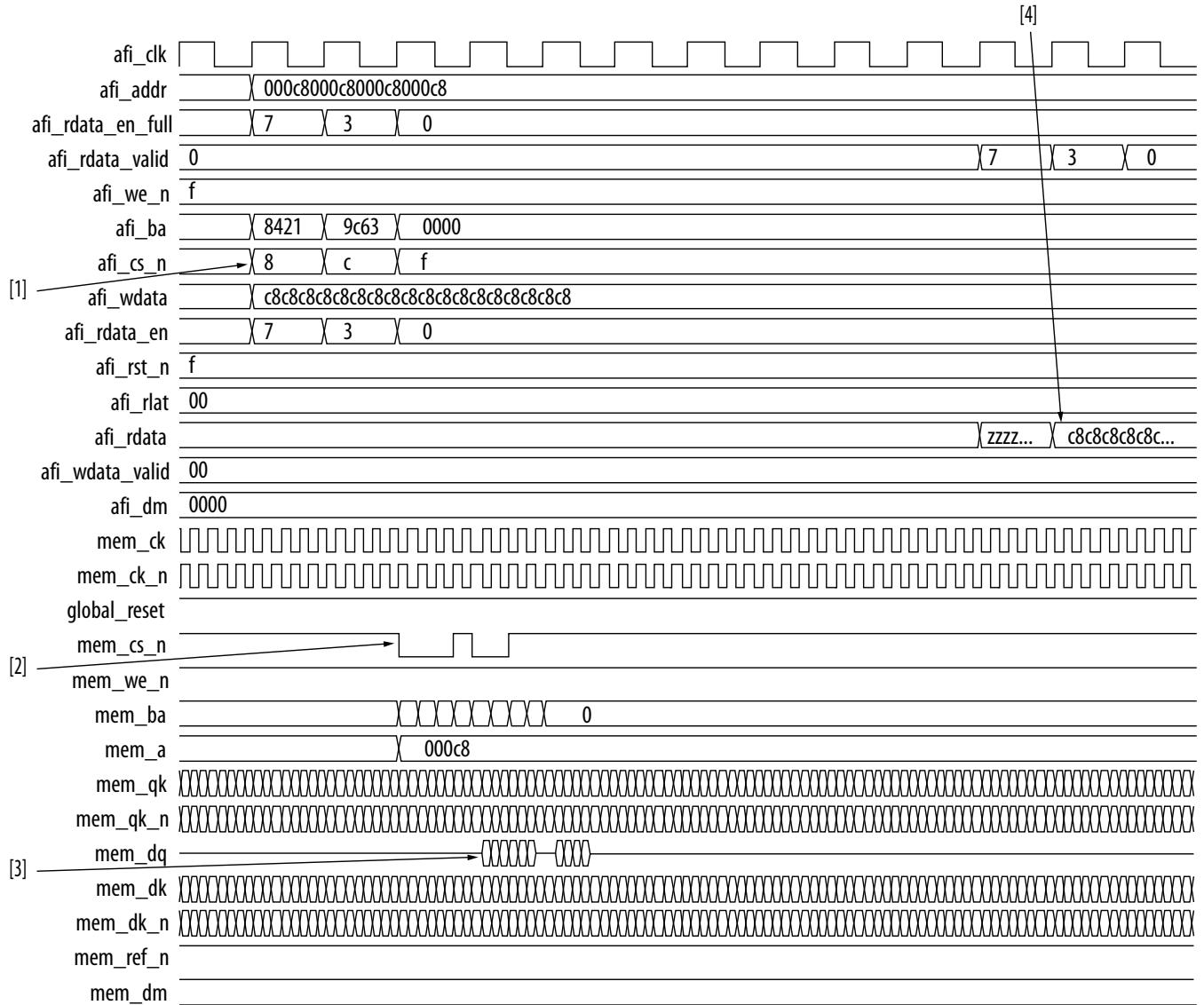
Notes for the above Figure:

1. Controller receives write command.
2. Controller receives write data.
3. Controller issues activate command to PHY.
4. PHY issues activate command to memory.
5. Controller issues write command to PHY.
6. PHY issues write command to memory.
7. Controller sends write data to PHY.
8. PHY sends write data to memory.

## 16.6. RLDRAM 3 Timing Diagrams

This topic contains timing diagrams for UniPHY-based external memory interface IP for RLDRAM 3 protocols.

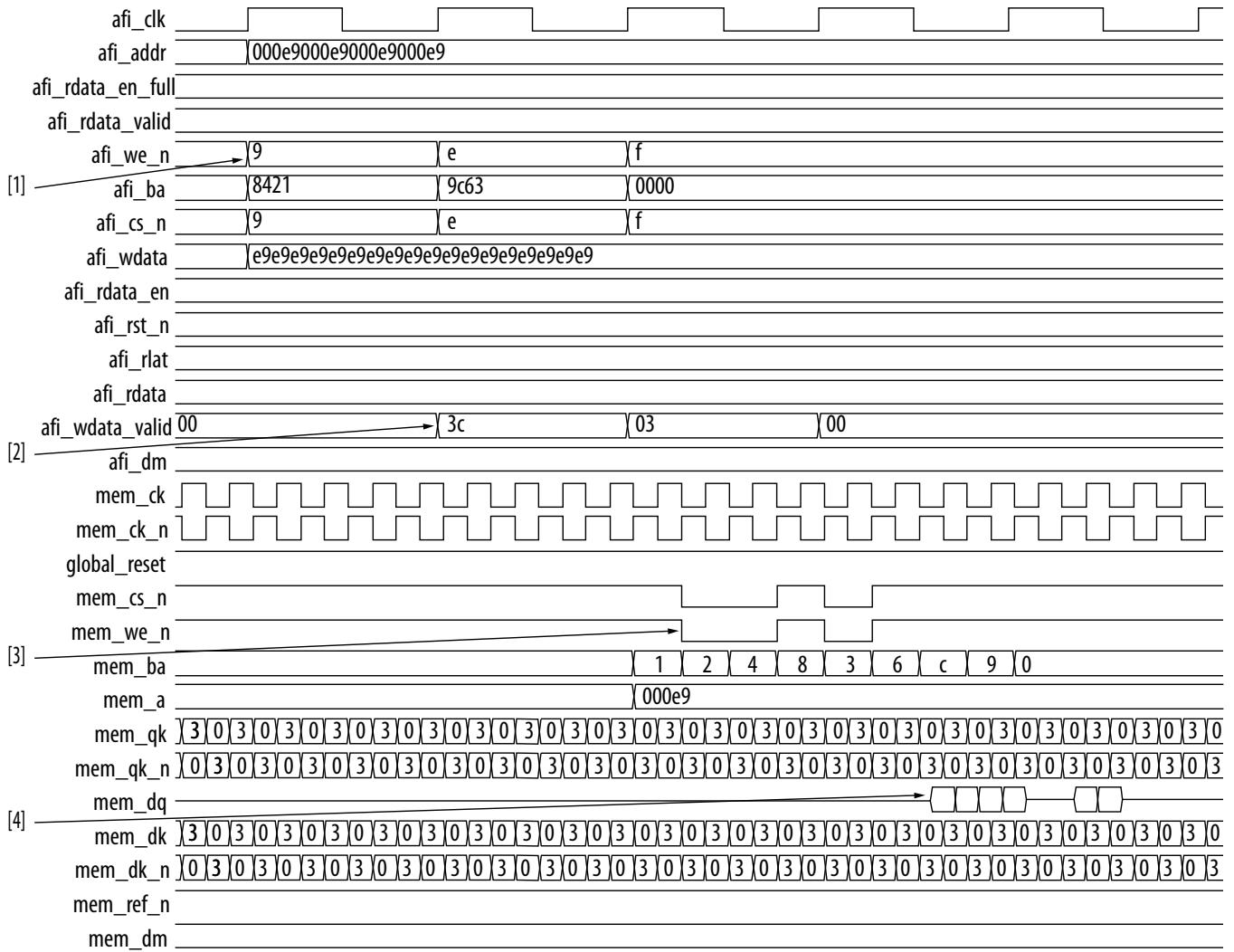
**Figure 203. Quarter-Rate RLDRAM 3 Read**



Notes for the above Figure:

1. Controller issues read command to PHY.
  2. PHY issues read command to memory.
  3. PHY receives data from memory.
  4. Controller receives read data from PHY.

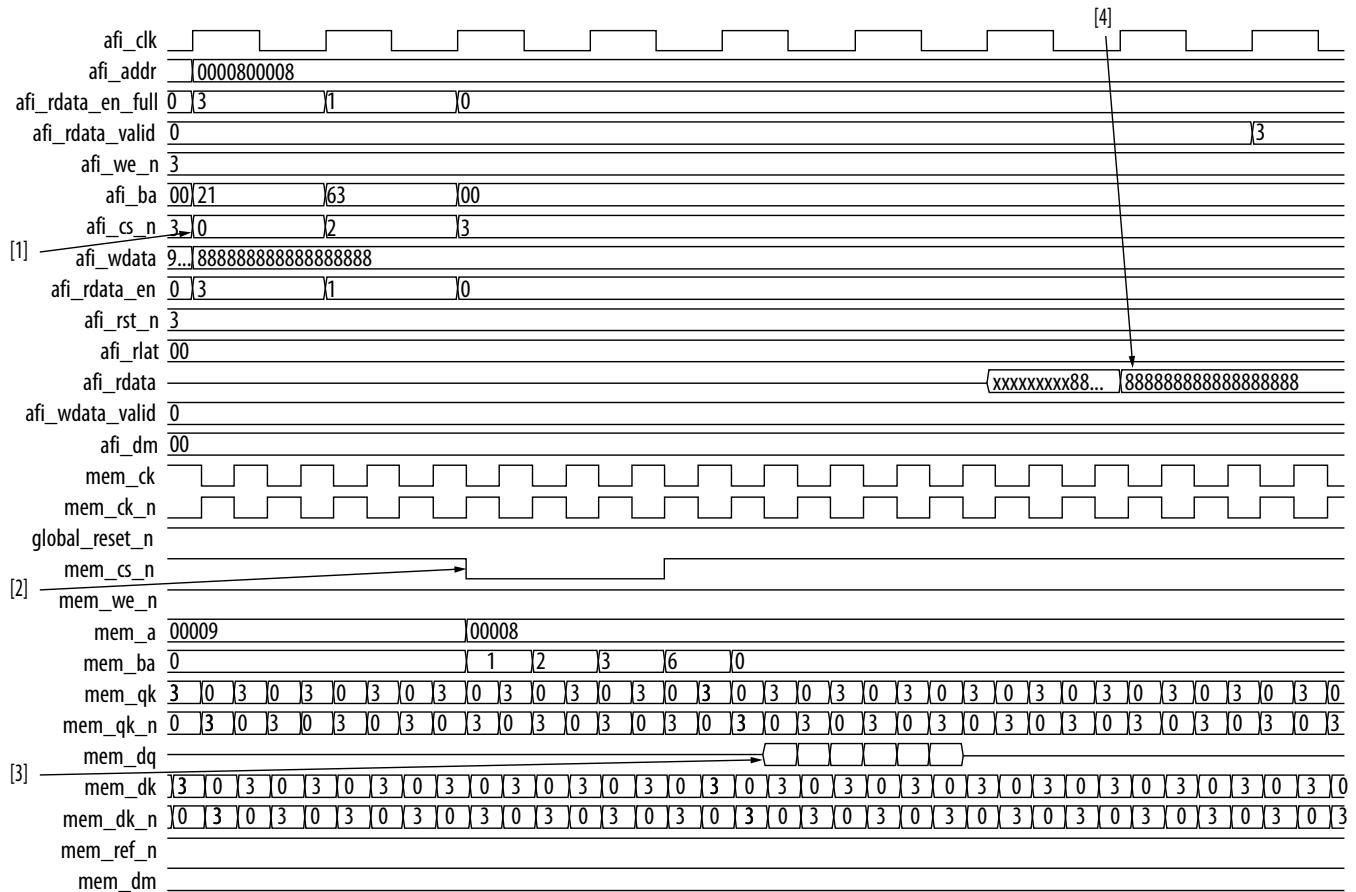
**Figure 204.** Quarter-Rate RLDRAM 3 Write



### Notes for the above Figure:

1. Controller issues write command to PHY.
  2. Data ready from controller for PHY.
  3. PHY issues write command to memory.
  4. PHY sends read data to memory.

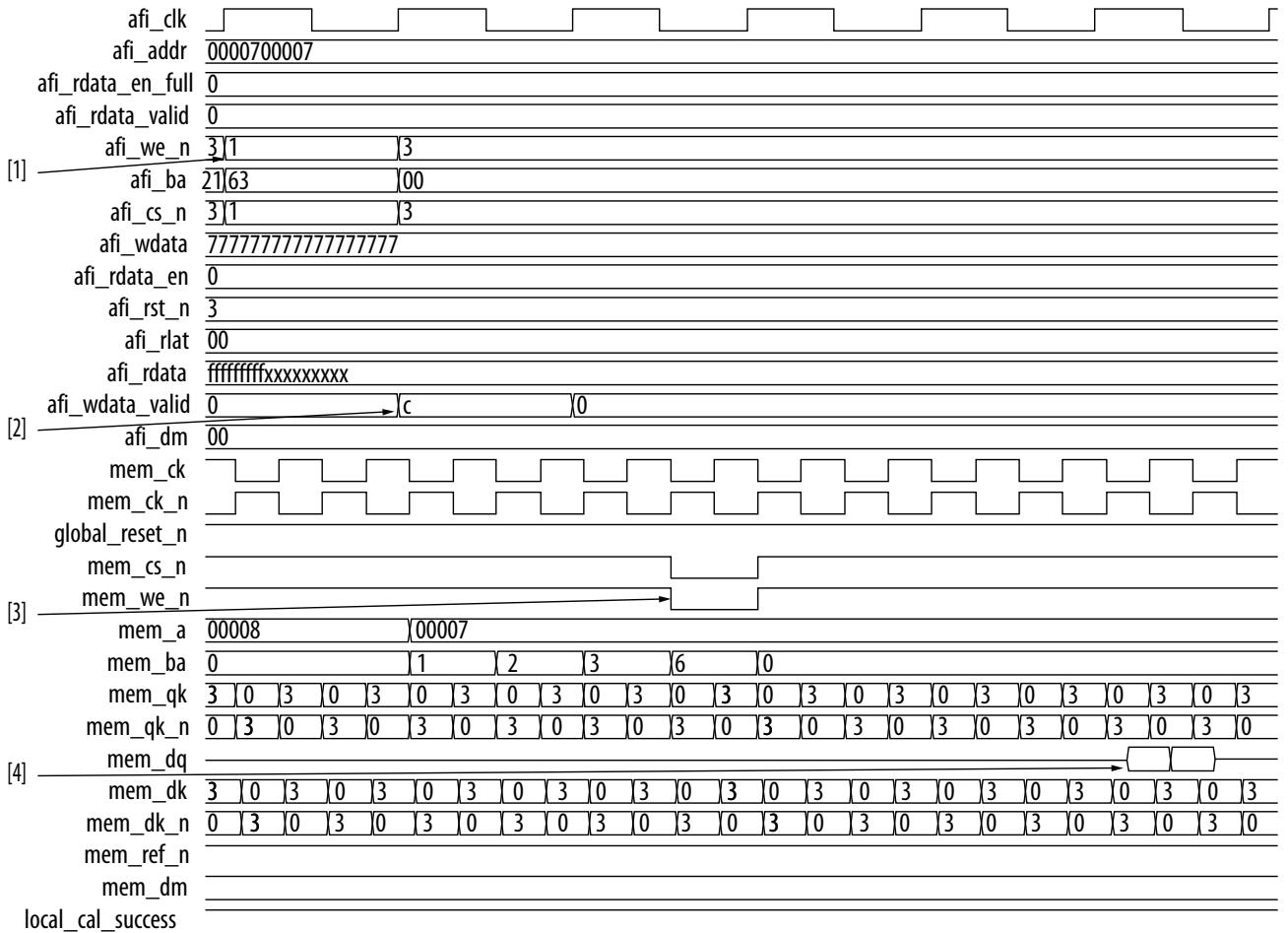
**Figure 205. Half-rate RLDRAM 3 Read**



Notes for the above Figure:

1. Controller issues read command to PHY.
  2. PHY issues read command to memory.
  3. PHY receives data from memory.
  4. Controller receives read data from PHY.

## Figure 206. Half-Rate RLDRAM 3 Write



### Notes for the above Figure:

1. Controller issues write command to PHY.
  2. Data ready from controller for PHY.
  3. PHY issues write command to memory.
  4. PHY sends read data to memory.

## 16.7. Document Revision History

| Date          | Version    | Changes              |
|---------------|------------|----------------------|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release. |
| May 2016      | 2016.05.02 | Maintenance release. |
| November 2015 | 2015.11.02 | Maintenance release. |
| May 2015      | 2015.05.04 | Maintenance release. |



| Date          | Version    | Changes  |
|---------------|------------|--|
| December 2014 | 2014.12.15 | Maintenance release.   |
| August 2014   | 2014.08.15 | Maintenance release.   |
| December 2013 | 2013.12.16 | Updated timing diagrams for LPDDR2.  |
| November 2012 | 2.1        | <ul style="list-style-type: none"><li>Added timing diagrams for RLDRAM 3.</li><li>Changed chapter number from 10 to 12.</li></ul>  |
| June 2012     | 2.0        | <ul style="list-style-type: none"><li>Added timing diagrams for LPDDR2.</li><li>Added Feedback icon.</li></ul>   |
| November 2011 | 1.1        | <ul style="list-style-type: none"><li>Consolidated timing diagrams from <i>11.0 DDR2 and DDR3 SDRAM Controller with UniPHY User Guide</i>, <i>QDR II and QDR II+ SRAM Controller with UniPHY User Guide</i>, and <i>RLDRAM II Controller with UniPHY IP User Guide</i>.</li><li>Added Read and Write diagrams for DDR3 quarter-rate.</li></ul> |



## 17. External Memory Interface Debug Toolkit

The EMIF Toolkit lets you run your own traffic patterns, diagnose and debug calibration problems, and produce margining reports for your external memory interface.

The toolkit is compatible with UniPHY-based external memory interfaces that use the Nios II-based sequencer, with toolkit communication enabled, and with Arria 10 EMIF IP. Toolkit communication is on by default in versions 10.1 and 11.0 of UniPHY IP; for version 11.1 and later, toolkit communication is on whenever debugging is enabled on the Diagnostics tab of the IP core interface.

The EMIF Toolkit can communicate with several different memory interfaces on the same device, but can communicate with only one memory device at a time.

*Note:* The EMIF Debug Toolkit does not support MAX 10 devices.

### 17.1. User Interface

The EMIF toolkit provides a graphical user interface for communication with connections.

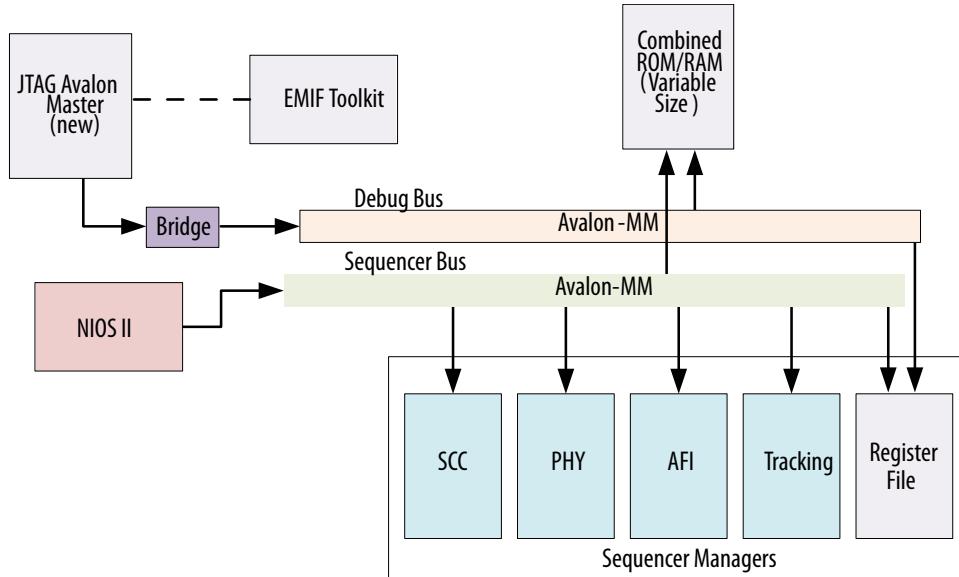
All functions provided in the toolkit are also available directly from the `quartus_sh` TCL shell, through the `external_memif_toolkit` TCL package. The availability of TCL support allows you to create scripts to run automatically from TCL. You can find information about specific TCL commands by running `help -pkg external_memif_toolkit` from the `quartus_sh` TCL shell.

If you want, you can begin interacting with the toolkit through the GUI, and later automate your workflow by creating TCL scripts. The toolkit GUI records a history of the commands that you run. You can see the command history on the History tab in the toolkit GUI.

#### 17.1.1. Communication

Communication between the EMIF Toolkit and external memory interface connections varies, depending on the connection type and version. In versions 10.1 and 11.0 of the EMIF IP, communication is achieved using direct communication to the Nios II-based sequencer. In version 11.1 and later, communication is achieved using a JTAG Avalon-MM master attached to the sequencer bus.

The following figure shows the structure of UniPHY-based IP version 11.1 and later, with JTAG Avalon-MM master attached to sequencer bus masters.

**Figure 207. UniPHY IP Version 11.1 and Later, with JTAG Avalon-MM Master**


### 17.1.2. Calibration and Report Generation

The EMIF Toolkit uses calibration differently, depending on the version of external memory interface in use. For versions 10.1 and 11.0 interfaces, the EMIF Toolkit causes the memory interface to calibrate several times, to produce the data from which the toolkit generates its reports. In version 11.1 and later, report data is generated during calibration, without need to repeat calibration. For version 11.1 and later, generated reports reflect the result of the previous calibration, without need to recalibrate unless you choose to do so.

## 17.2. Setup and Use

Before using the EMIF Toolkit, you should compile your design and program the target device with the resulting SRAM Object File (**.sof**). For designs compiled in the Quartus II software version 12.0 or earlier, debugging information is contained in the JTAG Debugging Information file (**.jdi**); however, for designs compiled in the Quartus II software version 12.1 or later, all debugging information resides in the **.sof** file.

You can run the toolkit using all your project files, or using only the Quartus Prime Project File (**.qpf**), Quartus Prime Settings File (**.qsf**), and **.sof** file; the **.jdi** file is also required for designs compiled prior to version 12.1. To ensure that all debugging information is correctly synchronized for designs compiled prior to version 12.1, ensure that the **.sof** and **.jdi** files that you used are generated during the same run of the Quartus Prime Assembler.

After you have programmed the target device, you can run the EMIF Toolkit and open your project. You can then use the toolkit to create connections to the external memory interface.



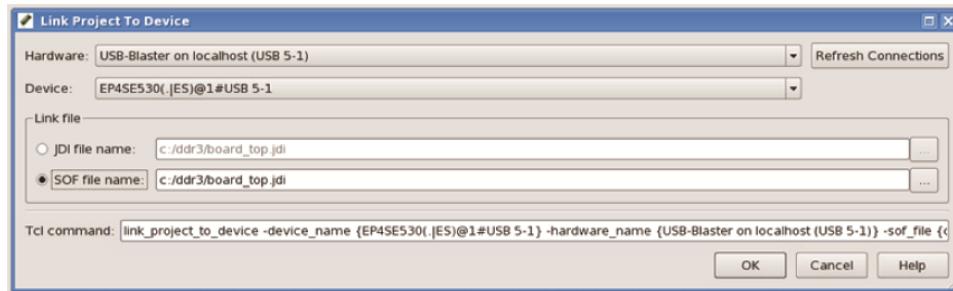
## 17.2.1. General Workflow

To use the EMIF Toolkit, you must link your compiled project to a device, and create a communication channel to the connection that you want to examine.

## 17.2.2. Linking the Project to a Device

1. To launch the toolkit, select External Memory Interface Toolkit from the Tools menu in the Quartus Prime software.
2. After you have launched the toolkit, open your project and click the **Initialize connections** task in the **Tasks** window, to initialize a list of all known connections.
3. To link your project to a specific device on specific hardware, perform the following steps:
  - a. Click the **Link Project to Device** task in the **Tasks** window.
  - b. Select the desired hardware from the **Hardware** dropdown menu in the **Link Project to Device** dialog box.
  - c. Select the desired device on the hardware from the **Device** dropdown menu in the **Link Project to Device** dialog box.
  - d. Select the correct **Link file type**, depending on the version of software in which your design was compiled:
    - If your design was compiled in the Quartus II software version 12.0 or earlier, select **JDI** as the **Link file type**, verify that the **.jdi** file is correct for your **.sof** file, and click **Ok**.
    - If your design was compiled in the Quartus II software version 12.1 or later, or the Quartus Prime software, select **SOF** as the **Link file type**, verify that the **.sof** file is correct for your programmed device, and click **Ok**.

**Figure 208. Link Project to Device Dialog Box**



When you link your project to the device, the toolkit verifies all connections on the device against the information in the JDI or SOF file, as appropriate. If the toolkit detects any mismatch between the JDI file and the device connections, an error message is displayed.

For designs compiled using the Quartus II software version 12.1 or later, or the Quartus Prime software, the SOF file contains a design hash to ensure the SOF file used to program the device matches the SOF file specified for linking to a project. If the hash does not match, an error message appears.

If the toolkit successfully verifies all connections, it then attempts to determine the connection type for each connection. Connections of a known type are listed in the Linked Connections report, and are available for the toolkit to use.

### 17.2.3. Establishing Communication to Connections

After you have completed linking the project, you can establish communication to the connections.

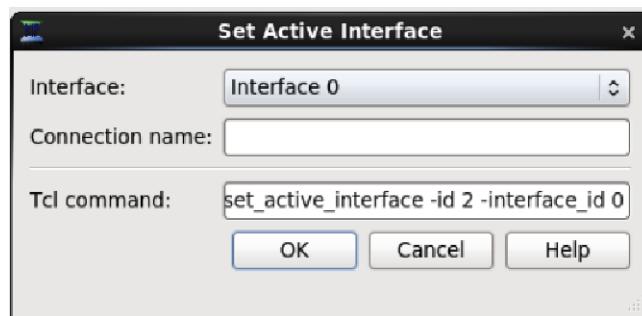
1. In the Tasks window,
  - Click **Establish Memory Interface Connection** to create a connection to the external memory interface.
  - Click **Establish Efficiency Monitor Connection** to create a connection to the efficiency monitor.
  - Click **Establish Traffic Generator Connection** to create a connection to the Traffic Generator 2.0 (if enabled, Arria 10 only).
2. To create a communication channel to a connection, select the desired connection from the displayed pulldown menu of connections, and click **Ok**. The toolkit establishes a communication channel to the connection, creates a report folder for the connection, and creates a folder of tasks for the connection.

*Note:* By default, the connection and the reports and tasks folders are named according to the hierarchy path of the connection. If you want, you can specify a different name for the connection and its folders.
3. You can run any of the tasks in the folder for the connection; any resulting reports appear in the reports folder for the connection.

### 17.2.4. Selecting an Active Interface

If you have more than one external memory interface in an Arria 10 I/O column, you can select one instance as the active interface for debugging.

1. To select one of multiple EMIF instances in an Arria 10 I/O column, use the **Set Active Interface** dialog box.



2. If you want to generate reports for the new active interface, you must first recalibrate the interface.



## 17.2.5. Reports

The toolkit can generate a variety of reports, including summary, calibration, and margining reports for external memory interface connections. To generate a supported type of report for a connection, you run the associated task in the tasks folder for that connection.

### Summary Report

The Summary Report provides an overview of the memory interface; it consists of the following tables:

- Summary table. Provides a high-level summary of calibration results. This table lists details about the connection, IP version, IP protocol, and basic calibration results, including calibration failures. This table also lists the estimated average read and write data valid windows, and the calibrated read and write latencies.
- Interface Details table. Provides details about the parameterization of the memory IP. This table allows you to verify that the parameters in use match the actual memory device in use.
- Groups Masked from Calibration table. Lists any groups that were masked from calibration when calibration occurred. Masked groups are ignored during calibration.  
*Note:* This table applies only to UniPHY-based interfaces; it is not applicable to Arria 10 EMIF.
- Ranks Masked from Calibration tables (DDR2 and DDR3 only). Lists any ranks that were masked from calibration when calibration occurred. Masked ranks are ignored during calibration.

### Calibration Report (UniPHY)

The Calibration Report provides detailed information about the margins observed before and after calibration, and the settings applied to the memory interface during calibration; it consists of the following tables:

- Per DQS Group Calibration table: Lists calibration results for each group. If a group fails calibration, this table also lists the reason for the failure.  
*Note:* If a group fails calibration, the calibration routine skips all remaining groups. You can deactivate this behaviour by running the **Enable Calibration for All Groups On Failure** command in the toolkit.
- DQ Pin Margins Observed Before Calibration table: Lists the DQ pin margins observed before calibration occurs. You can refer to this table to see the per-bit skews resulting from the specific silicon and board that you are using.
- DQS Group Margins Observed During Calibration table: Lists the DQS group margins observed during calibration.
- DQ Pin Settings After Calibration and DQS Group Settings After Calibration table: Lists the settings made to all dynamically controllable parts of the memory interface as a result of calibration. You can refer to this table to see the modifications made by the calibration algorithm.



### Calibration Report (Arria 10 EMIF)

The Calibration Report provides detailed information about the margins observed during calibration, and the settings applied to the memory interface during calibration; it consists of the following tables:

- Calibration Status Per Group table: Lists the pass/fail status per group.
- DQ Pin Margins Observed During Calibration table: Lists the DQ read/write margins and calibrated delay settings. These are the expected margins after calibration, based on calibration data patterns. This table also contains DM/DBI margins, if applicable.
- DQS Pin Margins Observed During Calibration table: Lists the DQS margins observed during calibration.
- FIFO Settings table: Lists the VFIFO and LFIFO settings made during calibration.
- Latency Observed During Calibration table: Lists the calibrated read/write latency.
- Address/Command Margins Observed During Calibration table: Lists the margins on calibrated A/C pins, for protocols that support Address/Command calibration.

### Margin Report

The Margin Report lists the post-calibration margins for each DQ and data mask pin, keeping all other pin settings constant; it consists of the following tables:

- DQ Pin Post Calibration Margins table. Lists the margin data in tabular format.
- Read Data Valid Windows report. Shows read data valid windows in graphical format.
- Write Data Valid Windows report. Shows write data valid windows in graphical format.

**Note:** The Margin Report applies only to UniPHY-based interfaces; it is not applicable to Arria 10 EMIF. For Arria 10 EMIF, the Calibration Report provides equivalent information.

## 17.3. Operational Considerations

Some features and considerations are of interest in particular situations.

### Specifying a Particular JDI File

Correct operation of the EMIF Toolkit depends on the correct JDI being used when linking the project to the device. The JDI file is produced by the Quartus Prime Assembler, and contains a list of all system level debug nodes and their heirarchy path names. If the default **.jdi** file name is incorrect for your project, you must specify the correct **.jdi** file. The **.jdi** file is supplied during the link-project-to-device step, where the *revision\_name.jdi* file in the project directory is used by default. To supply an alternative **.jdi** file, click on the ellipse then select the correct **.jdi** file.



### PLL Status

When connecting to DDR-based external memory interface connections, the PLL status appears in the Establish Connection dialog box when the IP is generated to use the CSR controller port, allowing you to immediately see whether the PLL status is locked. If the PLL is not locked, no communication can occur until the PLL becomes locked and the memory interface reset is deasserted.

When you are linking your project to a device, an error message will occur if the toolkit detects that a JTAG Avalon-MM master has no clock running. You can run the **Reindex Connections** task to have the toolkit rescan for connections and update the status and type of found connections in the Linked Connections report.

### Margining Reports

The EMIF Toolkit can display margining information showing the post-calibration data-valid windows for reads and writes. Margining information is determined by individually modifying the input and output delay chains for each data and strobe/clock pin to determine the working region. The toolkit can display margining data in both tabular and hierarchical formats.

### Group Masks

To aid in debugging your external memory interface, the EMIF Toolkit allows you to mask individual groups and ranks from calibration. Masked groups and ranks are skipped during the calibration process, meaning that only unmasked groups and ranks are included in calibration. Subsequent mask operations overwrite any previous masks.

### Using with Arria 10 Devices in a PHY-Only Configuration

If you want to use the Debug Toolkit with an Arria 10 EMIF IP in a PHY-only configuration, you must connect a debug clock and a reset signal to the Arria 10 external memory interface debug component. Intel recommends using the AFI clock and AFI reset for this purpose.

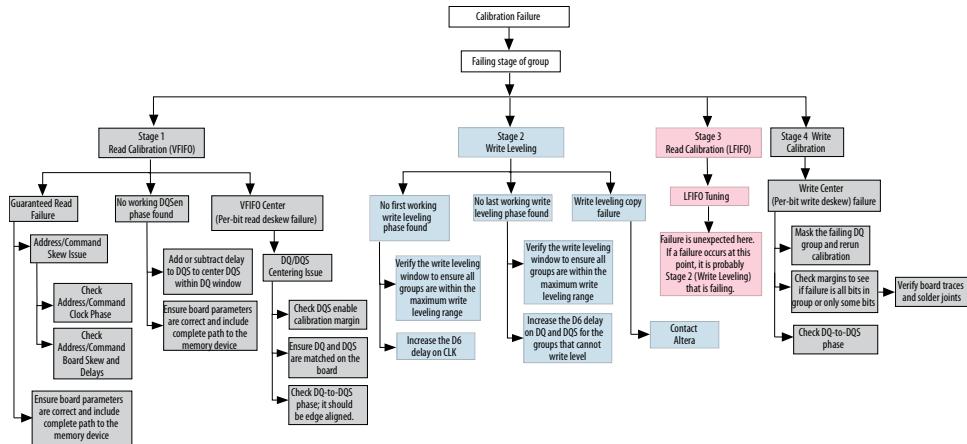
**Note:** For information about calibration stages in UniPHY-based interfaces, refer to *UniPHY Calibration Stages* in the *Functional Description - UniPHY* chapter. For information about calibration stages in Arria 10 EMIF IP, refer to *Arria 10 EMIF Calibration*, in the *Functional Description - Arria 10 EMIF* chapter.

### Related Information

- [Calibration Stages](#) on page 56
- [Functional Description—UniPHY](#) on page 13
- [Functional Description—Intel Arria 10 EMIF IP](#) on page 141

## 17.4. Troubleshooting

In the event of calibration failure, refer to the following figure to assist in troubleshooting your design. Calibration results and failing stages are available through the external memory interface toolkit.

**Figure 209. Debugging Tips**


## 17.5. Debug Report for Arria V and Cyclone V SoC Devices

The External Memory Interface Debug Toolkit and EMIF On-Chip Debug Port do not work with Arria V and Cyclone V SoC devices. Debugging information for Arria V and Cyclone V SoC devices is available by enabling a debug output report, which contains similar information.

### 17.5.1. Enabling the Debug Report for Arria V and Cyclone V SoC Devices

To enable a debug report for Arria V or Cyclone V SoC devices, perform the following steps:

1. Open the `<design_name>/hps_isw_handoff/sequencerDefines.h` file in a text editor.
2. In the `sequencerDefines.h` file, locate the following line: `#define RUNTIME_CAL_REPORT 0`
3. Change `#define RUNTIME_CAL_REPORT 0` to `#define RUNTIME_CAL_REPORT 1`, and save the file.
4. Generate the board support package (BSP) with semihosting enabled, or with UART output.

The system will now generate the debugging report as part of the calibration process.

### 17.5.2. Determining the Failing Calibration Stage for a Cyclone V or Arria V HPS SDRAM Controller

To determine the failing calibration stage, you must turn on the debug output report by setting the `RUNTIME_CAL_REPORT` option to 1 in the `sequencerDefines.h` file, located in the `hps_isw_handoff` directory.



If calibration fails, the following statements are printed in the debug output report:

```
SEQ.C: Calibration Failed
SEQ.C: Error Stage : <Num>
SEQ.C: Error Substage: <Num>
SEQ.C: Error Group : <Num>
```

To determine the stage and sub-stage, open the sequencer.h file in the hps\_isw\_handoff directory and look for the calibration defines:

```
/* calibration stages */
#define CAL_STAGE_NIL 0
#define CAL_STAGE_VFIFO 1
#define CAL_STAGE_WLEVEL 2
#define CAL_STAGE_LFIFO 3
#define CAL_STAGE_WRITES 4
#define CAL_STAGE_FULLTEST 5
#define CAL_STAGE_REFRESH 6
#define CAL_STAGE_CAL_SKIPPED 7
#define CAL_STAGE_CAL_ABORTED 8
#define CAL_STAGE_VFIFO_AFTER_WRITES 9
/* calibration substages */
#define CAL_SUBSTAGE_NIL 0
#define CAL_SUBSTAGE_GUARANTEED_READ 1
#define CAL_SUBSTAGE_DQS_EN_PHASE 2
#define CAL_SUBSTAGE_VFIFO_CENTER 3
#define CAL_SUBSTAGE_WORKING_DELAY 1
#define CAL_SUBSTAGE_LAST_WORKING_DELAY 2
#define CAL_SUBSTAGE_WLEVEL_COPY 3
#define CAL_SUBSTAGE_WRITES_CENTER 1
#define CAL_SUBSTAGE_READ_LATENCY 1
#define CAL_SUBSTAGE_REFRESH 1
```

For details about the stages of calibration, refer to *Calibration Stages in Functional Description - UniPHY*.

#### Related Information

[Calibration Stages](#) on page 56

## 17.6. On-Chip Debug Port for UniPHY-based EMIF IP

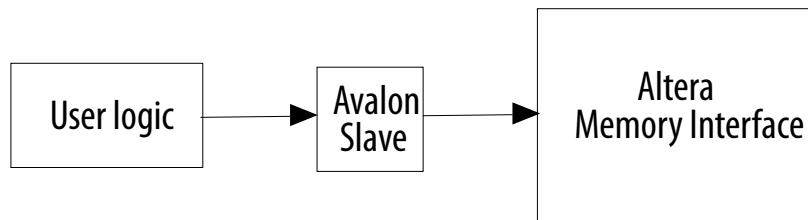
The EMIF On-Chip Debug Port allows user logic to access the same calibration data used by the EMIF Toolkit, and allows user logic to send commands to the sequencer. You can use the EMIF On-Chip Debug Port to access calibration data for your design and to send commands to the sequencer just as the EMIF Toolkit would. The following information is available:

- Pass/fail status for each DQS group
- Read and write data valid windows for each group

In addition, user logic can request the following commands from the sequencer:

- Destructive recalibration of all groups
- Masking of groups and ranks
- Generation of per-DQ pin margining data as part of calibration

The user logic communicates through an Avalon-MM slave interface as shown below.

**Figure 210. User Logic Access**

### 17.6.1. Access Protocol

The On-Chip Debug Port provides access to calibration data through an Avalon-MM slave interface. To send a command to the sequencer, user logic sends a command code to the command space in sequencer memory. The sequencer polls the command space for new commands after each group completes calibration, and continuously after overall calibration has completed.

The communication protocol to send commands from user logic to the sequencer uses a multistep handshake with a data structure as shown below, and an algorithm as shown in the figure which follows.

```
typedef struct_debug_data_struct {  
    ...  
    // Command interaction  
    alt_u32 requested_command;  
    alt_u32 command_status;  
    alt_u32 command_parameters[COMMAND_PARAM_WORDS]; ...  
}
```

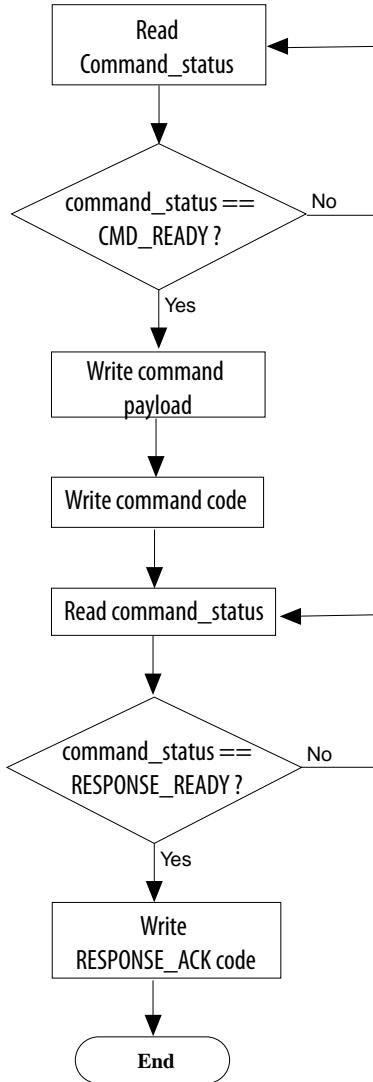
To send a command to the sequencer, user logic must first poll the `command_status` word for a value of `TCLDBG_TX_STATUS_CMD_READY`, which indicates that the sequencer is ready to accept commands. When the sequencer is ready to accept commands, user logic must write the command parameters into `command_parameters`, and then write the command code into `requested_command`.

The sequencer detects the command code and replaces `command_status` with `TCLDBG_TX_STATUS_CMD_EXE`, to indicate that it is processing the command. When the sequencer has finished running the command, it sets `command_status` to `TCLDBG_TX_STATUS_RESPONSE_READY` to indicate that the result of the command is available to be read. (If the sequencer rejects the requested command as illegal, it sets `command_status` to `TCLDBG_TX_STATUS_ILLEGAL_CMD`.)

User logic acknowledges completion of the command by writing `TCLDBG_CMD_RESPONSE_ACK` to `requested_command`. The sequencer responds by setting `command_status` back to `STATUS_CMD_READY`. (If an illegal command is received, it must be cleared using `CMD_RESPONSE_ACK`.)



**Figure 211. Debugging Algorithm Flowchart**



### 17.6.2. Command Codes Reference

The following table lists the supported command codes for the On-Chip Debug Port.

**Table 136. Supported Command Codes**

| Command                             | Parameters    | Description  |
|-------------------------------------|---------------|--|
| TCLDBG_RUN_MEM_CALIBRATE            | None          | Runs the calibration routine.                          |
| TCLDBG_MARK_ALL_DQS_GROUPS_AS_VALID | None          | Marks all groups as valid for calibration.             |
| TCLDBG_MARK_GROUP_AS_SKIP           | Group to skip | Mark the specified group to be skipped by calibration. |
| <i>continued...</i>                 |               |  |



| Command                        | Parameters   | Description   |
|--------------------------------|--------------|---|
| TCLDBG_MARK_ALL_RANKS_AS_VALID | None         | Mark all ranks as valid for calibration               |
| TCLDBG_MARK_RANK_AS_SKIP       | Rank to skip | Mark the specified rank to be skipped by calibration. |
| TCLDBG_ENABLE_MARGIN_REPORT    | None         | Enables generation of the margin report.              |

### 17.6.3. Header Files

The external memory interface IP generates header files which identify the debug data structures and memory locations used with the EMIF On-Chip Debug Port. You should refer to these header files for information required for use with your core user logic. It is highly recommended to use a software component (such as a Nios II processor) to access the calibration debug data.

The header files are unique to your IP parameterization and version, therefore you must ensure that you are referring to the correct version of header for your design. The names of the header files are: **core\_debug.h** and **core\_debugDefines.h**. The header files reside in `<design_name>/<design_name>_s0_software`.

### 17.6.4. Generating IP With the Debug Port

The following steps summarize the procedure for implementing your IP with the EMIF On-Chip Debug Port enabled.

1. Start the Quartus Prime software and generate a new external memory interface. For QDR II and RLDRAM II protocols, ensure that sequencer optimization is set to **Performance** (for Nios II-based sequencer).
2. On the **Diagnostics** tab of the parameter editor, turn on **Enable EMIF On-Chip Debug Port**.
3. Ensure that the **EMIF On-Chip Debug Port interface type** is set to **Avalon-MM Slave**.
4. Click **Finish** to generate your IP.
5. Find the Avalon interface in the top-level generated file. Connect this interface to your debug component.

```
input wire [19:0] seq_debug_addr,      //      seq_debug.address
input wire          seq_debug_read_req, //      .read
output wire [31:0] seq_debug_rdata,   //      .readdata
input wire          seq_debug_write_req, //      .write
input wire [31:0] seq_debug_wdata,    //      .writedata
output wire          seq_debug_waitrequest, //      .waitrequest
input wire [3:0] seq_debug_be,        //      .byteenable
output wire          seq_debug_rdata_valid //      .readdatavalid
```

If you are using UniPHY-based IP with the hard memory controller, also connect the `seq_debug_clk` and `seq_debug_reset_in` signals to clock and asynchronous reset signals that control your debug logic.

6. Find the **core\_debug.h** and **core\_debugDefines.h** header files in `<design_name>/<design_name>_s0_software` and include these files in your debug component code.
7. Write your debug component using the supported command codes, to read and write to the Avalon-MM interface.



The debug data structure resides at the memory address SEQ\_CORE\_DEBUG\_BASE, which is defined in the **core\_debugDefines.h** header file.

### 17.6.5. Example C Code for Accessing Debug Data

A typical use of the EMIF On-Chip Debug Port might be to recalibrate the external memory interface, and then access the reports directly using the `summary_report_ptr`, `cal_report_ptr`, and `margin_report_ptr` pointers, which are part of the debug data structure.

The following code sample illustrates:

```
/*
 * DDR3 UniPHY sequencer core access example
 */

#include <stdio.h>
#include <unistd.h>
#include <io.h>
#include "core_debugDefines.h"

int send_command(volatile debug_data_t* debug_data_ptr, int command, int args[], int num_args)
{
    volatile int i, response;

    // Wait until command_status is ready
    do {
        response = IORD_32DIRECT(&(debug_data_ptr->command_status), 0);
    } while(response != TCLDBG_TX_STATUS_CMD_READY);

    // Load arguments
    if(num_args > COMMAND_PARAM_WORDS)
    {
        // Too many arguments
        return 0;
    }
    for(i = 0; i < num_args; i++)
    {
        IOWR_32DIRECT(&(debug_data_ptr->command_parameters[i]), 0, args[i]);
    }
    // Send command code
    IOWR_32DIRECT(&(debug_data_ptr->requested_command), 0, command);
    // Wait for acknowledgment
    do {
        response = IORD_32DIRECT(&(debug_data_ptr->command_status), 0);
    } while(response != TCLDBG_TX_STATUS_RESPONSE_READY && response != TCLDBG_TX_STATUS_ILLEGAL_CMD);
    // Acknowledge response
    IOWR_32DIRECT(&(debug_data_ptr->requested_command), 0,
TCLDBG_CMD_RESPONSE_ACK);
    // Return 1 on success, 0 on illegal command
    return (response != TCLDBG_TX_STATUS_ILLEGAL_CMD);
}
int main()
{
    volatile debug_data_t* my_debug_data_ptr;
    volatile debug_summary_report_t* my_summary_report_ptr;
    volatile debug_cal_report_t* my_cal_report_ptr;
    volatile debug_margin_report_t* my_margin_report_ptr;
    volatile debug_cal_observed_dq_margins_t* cal_observed_dq_margins_ptr;
    int i, j, size;
    int args[COMMAND_PARAM_WORDS];
    // Initialize pointers to the debug reports
    my_debug_data_ptr = (debug_data_t*)SEQ_CORE_DEBUG_BASE;
    my_summary_report_ptr = (debug_summary_report_t*)
(IORD_32DIRECT(&(my_debug_data_ptr->summary_report_ptr), 0));
```

```
my_cal_report_ptr = (debug_cal_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr->cal_report_ptr), 0));
my_margin_report_ptr = (debug_margin_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr->margin_report_ptr), 0));

// Activate all groups and ranks
send_command(my_debug_data_ptr, TCLDBG_MARK_ALL_DQS_GROUPS_AS_VALID, 0, 0);
send_command(my_debug_data_ptr, TCLDBG_MARK_ALL_RANKS_AS_VALID, 0, 0);
send_command(my_debug_data_ptr, TCLDBG_ENABLE_MARGIN_REPORT, 0, 0);

// Mask group 4
args[0] = 4;
send_command(my_debug_data_ptr, TCLDBG_MARK_GROUP_AS_SKIP, args, 1);
send_command(my_debug_data_ptr, TCLDBG_RUN_MEM_CALIBRATE, 0, 0);

// SUMMARY
printf("SUMMARY REPORT\n");
printf("mem_address_width: %u\n", IORD_32DIRECT(&(my_summary_report_ptr->mem_address_width), 0));
printf("mem_bank_width: %u\n", IORD_32DIRECT(&(my_summary_report_ptr->mem_bank_width), 0));
// etc...

// CAL REPORT
printf("CALIBRATION REPORT\n");
// DQ read margins
for(i = 0; i < RW_MGR_MEM_DATA_WIDTH; i++)
{
    cal_observed_dq_margins_ptr = &(my_cal_report_ptr->cal_dq_in_margins[i]);
    printf("0x%x DQ %d Read Margin (taps): -%d : %d\n", (unsigned
int)cal_observed_dq_margins_ptr, i,
        IORD_32DIRECT(&(cal_observed_dq_margins_ptr->left_edge), 0),
        IORD_32DIRECT(&(cal_observed_dq_margins_ptr->right_edge), 0));
}
// etc...
return 0;
}
```

## 17.7. On-Chip Debug Port for Arria 10 EMIF IP

The EMIF On-Chip Debug Port allows user logic to access the same calibration data used by the EMIF Toolkit, and allows user logic to send commands to the sequencer. You can use the EMIF On-Chip Debug Port to access calibration data for your design and to send commands to the sequencer just as the EMIF Toolkit would. The following information is available:

- Pass/fail status for each DQS group
- Read and write data valid windows for each group

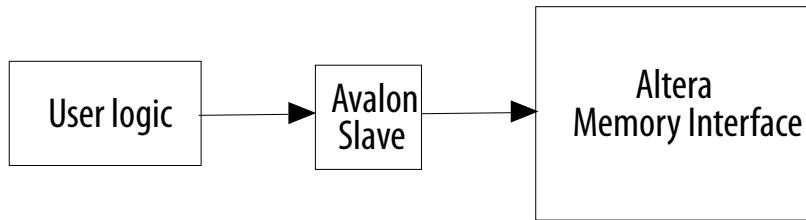
In addition, user logic can request the following commands from the sequencer:

- Destructive recalibration of all groups
- Masking of groups and ranks
- Generation of per-DQ pin margining data as part of calibration

The user logic communicates through an Avalon-MM slave interface as shown below.



Figure 212. User Logic Access



### 17.7.1. Access Protocol

The On-Chip Debug Port provides access to calibration data through an Avalon-MM slave interface. To send a command to the sequencer, user logic sends a command code to the command space in sequencer memory. The sequencer polls the command space for new commands after each group completes calibration, and continuously after overall calibration has completed.

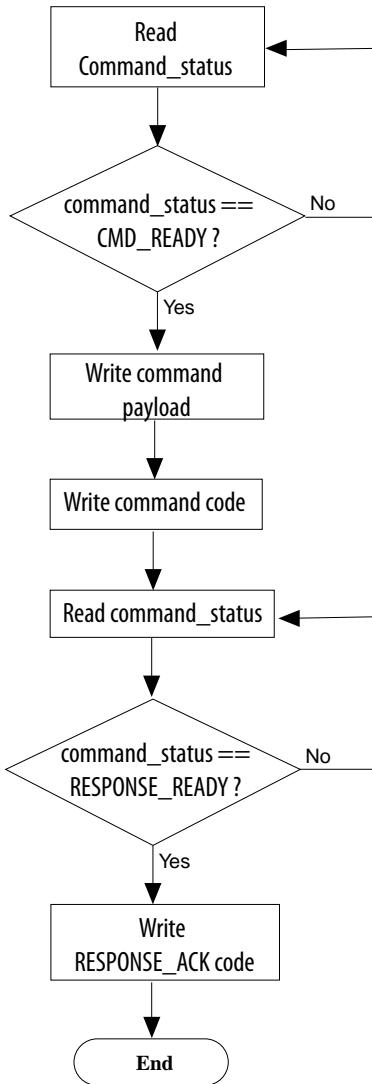
The communication protocol to send commands from user logic to the sequencer uses a multistep handshake with a data structure as shown below, and an algorithm as shown in the figure which follows.

```
typedef struct_debug_data_struct {  
    ...  
    // Command interaction  
    alt_u32 requested_command;  
    alt_u32 command_status;  
    alt_u32 command_parameters[COMMAND_PARAM_WORDS];...  
}
```

To send a command to the sequencer, user logic must first poll the `command_status` word for a value of `TCLDBG_TX_STATUS_CMD_READY`, which indicates that the sequencer is ready to accept commands. When the sequencer is ready to accept commands, user logic must write the command parameters into `command_parameters`, and then write the command code into `requested_command`.

The sequencer detects the command code and replaces `command_status` with `TCLDBG_TX_STATUS_CMD_EXE`, to indicate that it is processing the command. When the sequencer has finished running the command, it sets `command_status` to `TCLDBG_TX_STATUS_RESPONSE_READY` to indicate that the result of the command is available to be read. (If the sequencer rejects the requested command as illegal, it sets `command_status` to `TCLDBG_TX_STATUS_ILLEGAL_CMD`.)

User logic acknowledges completion of the command by writing `TCLDBG_CMD_RESPONSE_ACK` to `requested_command`. The sequencer responds by setting `command_status` back to `STATUS_CMD_READY`. (If an illegal command is received, it must be cleared using `CMD_RESPONSE_ACK`.)

**Figure 213. Debugging Algorithm Flowchart**


### 17.7.2. EMIF On-Chip Debug Port

In Arria 10 and later families, access to on-chip debug is provided through software running on a Nios processor connected to the external memory interface.

If you enable the Use Soft Nios Processor for On-Chip Debug option, the system instantiates a soft Nios processor, and software files are provided as part of the EMIF IP.

Instructions on how to use the software are available in the following file: :  
 <variation\_name>/altera\_emif\_arch\_nf\_<version\_number>/<synth|sim>/<variation\_name>\_altera\_emif\_arch\_nf\_<version number>\_<unique ID>\_readme.txt.



### 17.7.3. On-Die Termination Calibration

The **Calibrate Termination** feature lets you determine the optimal **On-Die Termination** and **Output Drive Strength** settings for your memory interface, for Arria 10 and later families.

The **Calibrate Termination** function runs calibration with all available termination settings and selects the optimal settings based on the calibration margins.

The **Calibrate Termination** feature is available for DDR3, DDR4, and RLDRAM 3 protocols, on Arria 10 devices.

### 17.7.4. Eye Diagram

The **Generate Eye Diagram** feature allows you to create read and write eye diagrams for each pin in your memory interface, for Arria 10 and later families.

The **Generate Eye Diagram** feature uses calibration data patterns to determine margins at each  $V_{ref}$  setting on both the FPGA pins and the memory device pins. A full calibration is done for each Vref setting. Other settings, such as DQ delay chains, will change for each calibration. At the end of a Generate Eye Diagram command, a default calibration is run to restore original behavior.

The **Generate Eye Diagram** feature is available for DDR4 and QDR-IV protocols, on Arria 10 devices.

## 17.8. Driver Margining for Arria 10 EMIF IP

The Driver Margining feature lets you measure margins on your memory interface using a driver with arbitrary traffic patterns.

Margins measured with this feature may differ from margins measured during calibration, because of different traffic patterns. Driver margining is not available if ECC is enabled.

To use driver margining, ensure that the following signals on the driver are connected to In-System Sources/Probes:

- Reset\_n: An active low reset signal
- Pass: A signal which indicates that the driver test has completed successfully. No further memory transactions must be sent after this signal is asserted.
- Fail: A signal which indicates that the driver test has failed. No further memory transactions must be sent after this signal is asserted.
- PNF (Pass Not Fail): An array of signals that indicate the pass/fail status of individual bits of a data burst. The PNF should be arranged such that each bit index corresponds to  $(\text{Bit of burst} * \text{DQ width}) + (\text{DQ pin})$ . A 1 indicates pass, 0 indicates fail. If the PNF width exceeds the capacity of one In-System Probe, specify them in PNF[1] and PNF[2]; otherwise, leave them blank.

If you are using the example design for EMIF, the In-System Sources/Probes can be enabled by adding the following line to your .qsf file:

```
set_global_assignment -name VERILOG_MACRO  
"ALTERA_EMIF_ENABLE_ISSP=1"
```



### 17.8.1. Determining Margin

The Driver Margining feature lets you measure margins on your Arria 10 EMIF IP interface using a driver with arbitrary traffic patterns.

The Driver Margining feature is available only for DDR3 and DDR4 interfaces on Arria 10 devices, when ECC is not enabled.

1. Establish a connection to the desired interface and ensure that it has calibrated successfully.
2. Select **Driver Margining** from the **Commands** folder under the target interface connection.
3. Select the appropriate **In-System Sources/Probes** using the drop-down menus.
4. If required, set additional options in the **Advanced Options** section:
  - Specify **Traffic Generator 2.0** to allow margining on a per-rank basis. Otherwise, margining is performed on all ranks together.
  - **Step size** specifies the granularity of the driver margining process. Larger step sizes allow faster margining but reduced accuracy. It is recommended to omit this setting.
  - **Adjust delays after margining** causes delay settings to be adjusted to the center of the window based on driver margining results.
  - The **Margin Read**, **Write**, **Write DM**, and **DBI** checkboxes allow you to control which settings are tested during driver margining. You can uncheck boxes to allow driver margining to complete more quickly.
5. Click **OK** to run the tests.  
The toolkit measures margins for **DQ** read/write and **DM**. The process may take several minutes, depending on the margin size and the duration of the driver tests. The test results are available in the *Margin Report*.

## 17.9. Read Setting and Apply Setting Commands for Arria 10 EMIF IP

The **Read Setting** command allows you to read calibration settings directly from the EMIF PHY. The **Apply Setting** command allows you to write calibration settings, to override existing settings for testing purposes.

### 17.9.1. Reading or Applying Calibration Settings

The **Read Setting** and **Apply Setting** commands let you read and write calibration settings directly.

The **Read Setting** and **Apply Setting** commands are available only for DDR3 and DDR4 interfaces on Arria 10 devices.

1. Establish a connection to the desired interface.
2. Select **Read Setting** or **Apply Setting** from the **Settings** folder under the target interface connection.
3. Select the desired setting type.



4. Select the desired rank shadow register to modify. (Leave this field blank if it is not applicable).
5. Select the index of the pin or group to modify.
6. (For the **Apply Setting** command) Enter the new value to apply to the desired location.
7. Click **OK**.  
The setting is read (or applied) using a read/write at the address indicated in the Tcl command window. You can perform similar transactions using the On-Chip Debug Port.

## 17.10. Traffic Generator 2.0

The Traffic Generator 2.0 lets you emulate traffic to the external memory, and helps you test, debug, and understand the performance of your external memory interface on hardware in a standalone fashion, without having to incorporate your entire design.

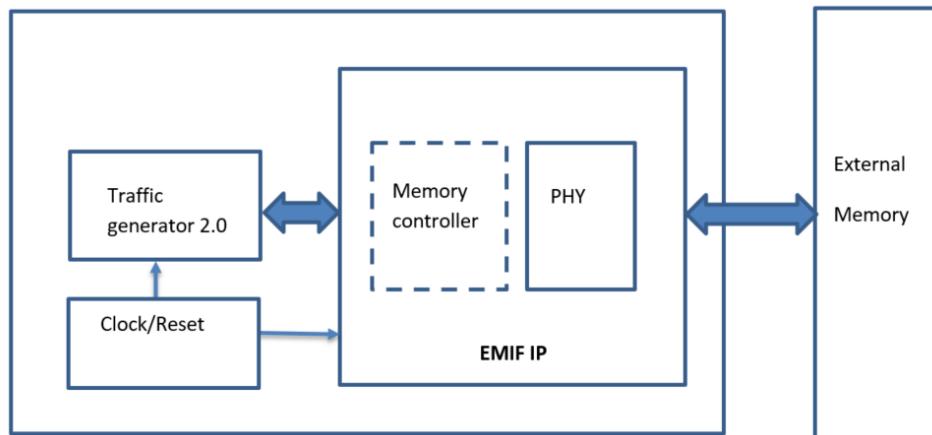
The Traffic Generator 2.0 lets you customize data patterns being written to the memory, address locations accessed in the memory, and the order of write and read transactions. You can use the traffic generator code with any FPGA architecture and memory protocol.

### 17.10.1. Configuring the Traffic Generator 2.0

The traffic generator replaces user logic to generate traffic to the external memory. You must incorporate the traffic generator design into the EMIF IP design during IP generation.

When you generate the example design in the parameter editor, the traffic generator module and EMIF IP are generated together. If you have an example design with the Traffic Generator 2.0 enabled, you can configure the traffic pattern using the EMIF Debug Toolkit.

**Figure 214. Traffic Generator 2.0 Generated with EMIF IP in Example Design Mode**





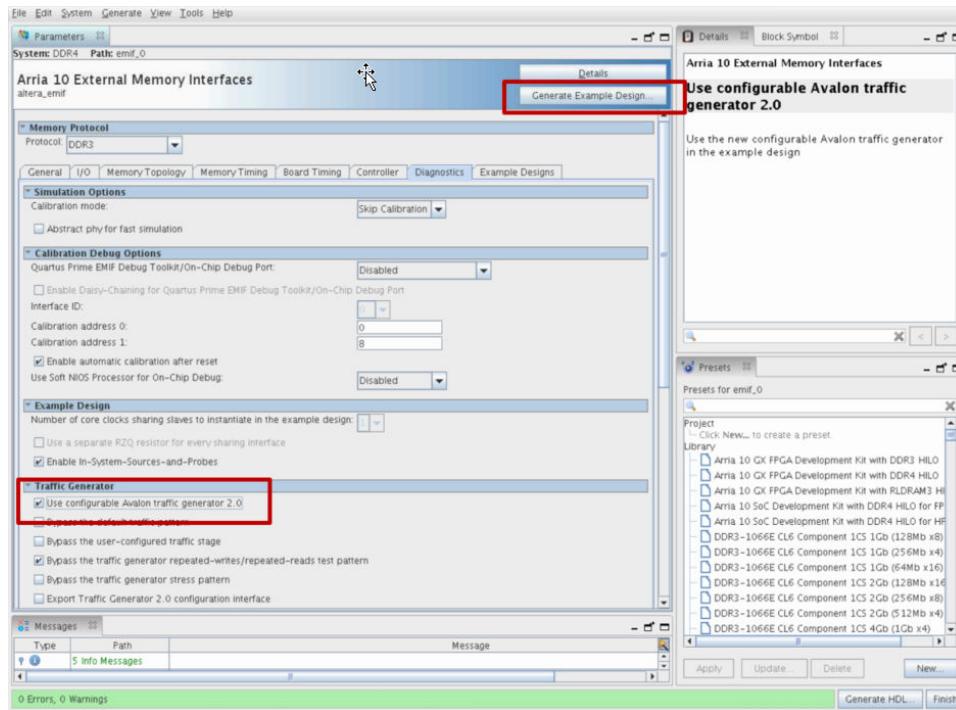
## Generating the External Memory Interface

1. Select the FPGA and Memory parameters.
2. On the Diagnostics tab, configure the following parameters:
  - a. Select **Use Configurable Avalon Traffic Generator 2.0**.
  - b. Configure the desired traffic pattern, by specifying traffic patterns to be bypassed. The traffic pattern not bypassed is issued to the memory immediately after completion of calibration. You can choose to bypass any of the following traffic patterns:
    - **Bypass the default traffic pattern** Specifies not to use the default traffic patterns from the traffic generator. The default patterns include single read/write, byte-enabled read/write, and block read/write.
    - **Bypass the user-configured traffic stage**. Specifies to skip the stage that uses the user-defined test bench file to configure the traffic generator in simulation.
    - **Bypass the traffic generator repeated-writes/repeated-reads test pattern**. Bypasses the traffic generator's repeat test stage, which causes every write and read to be repeated several times.
    - **Bypass the traffic generator stress pattern**. Bypasses a test stage intended to stress-test signal integrity and memory interface calibration.
    - **Export Traffic Generator 2.0 configuration interface**. Instantiates a port for traffic generator configuration. Use this port if the traffic generator is to be configured by user logic.
3. Click **Generate Example Design** to generate the EMIF IP, including the Traffic Generator 2.0 design, with the traffic pattern that you have configured.

*Note:* If you click the **Generate HDL** option instead, the Traffic Generator 2.0 design is not included in the generated IP.



Figure 215. Enabling the Traffic Generator 2.0 in the Parameter Editor



### 17.10.2. Running the Traffic Generator 2.0

You can use the EMIF Debug Toolkit to configure the traffic generator infrastructure to send custom traffic patterns to the memory.

1. Launch the EMIF Debug Toolkit by selecting **Tools > System Debugging Tools > External Memory Interface Toolkit**.
2. After you launch the toolkit, you must establish the following connections, before running the custom traffic generator:
  - Initialize Connections
  - Link Project to Device
  - Connections
    - Create Memory Interface Connection
    - Create Traffic Generator Connection
3. Launch the Traffic Generator by selecting **Traffic Generator > Settings > Run Custom Traffic Pattern**.

### 17.10.3. Understanding the Custom Traffic Generator User Interface

The Custom Traffic Generator interface lets you configure data patterns, the bytes to be enabled, the addressing mode, and the order in which traffic is organized.

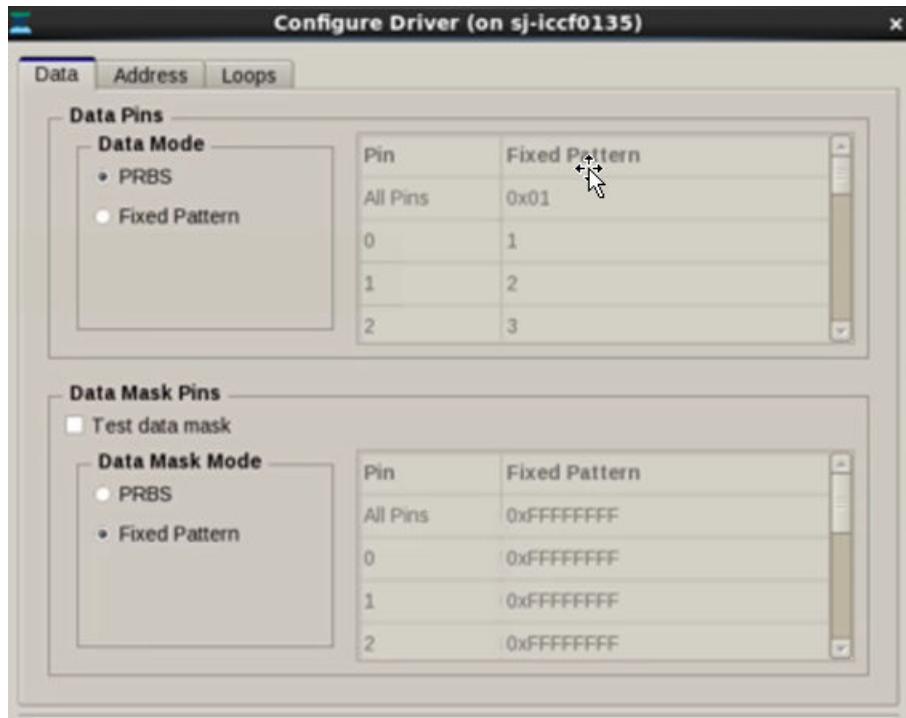
The interface has three tabs:

- **Data** tab
- **Address** tab
- **Loops** tab

### Data Tab

The **Data** tab is divided into **Data Pins** and **Data Mask Pins** sections.

**Figure 216. Data Tab**



The **Data Pins** section helps with customizing the patterns selected for the data pins. You can choose between two options for **Data Mode**:

- **PRBS** The default write data to all the data pins.
- **Fixed Pattern** Lets you specify a pattern to be written to the memory.

Select the **All Pins** option when you want to write the same data pattern to all the data pins. If data must be individually assigned to the data pins, you must enter the data value for each individual pin. The width of the data entered is based on the AVL\_TO\_DQ\_WIDTH\_RATIO, which is based on the ratio of the memory clock to the user clock.

All data bytes are enabled by default; the **Data Mask Pins** section lets you disable any of the bytes if you want to. To disable data bytes individually, select **Test data mask**.



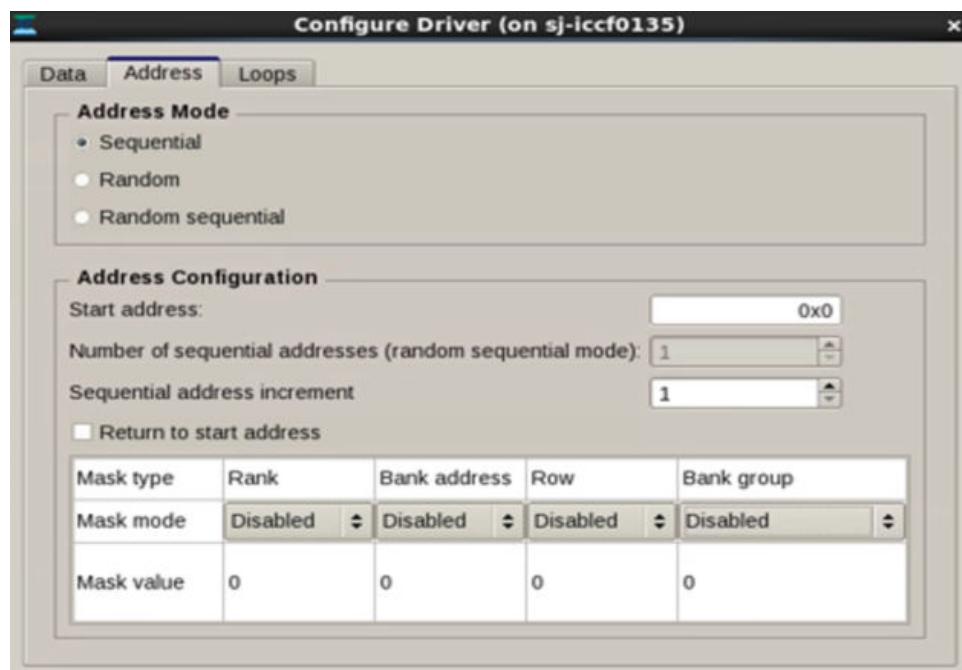
You can choose between two options for **Data Mode**:

- **PRBS** Specifies the PRBS pattern to enable or disable data bytes. A 1 denotes a data byte enabled, while a 0 denotes a data byte being masked or disabled.
- **Fixed Pattern** Lets you enable or disable individual bytes. You can apply byte enables to all pins or to individual bytes. A 1 denotes a data byte enabled, while a 0 denotes a data byte being masked or disabled.

### Address Tab

The **Address** tab lets you configure sequential, random, or random sequential (where the initial start address is random, but sequential thereafter) addressing schemes. The **Address** tab is divided into **Address Mode** and **Address Configuration** sections.

**Figure 217. Address Tab**



The **Address Mode** section lets you specify the pattern of addresses generated to access the memory. You can choose between three address modes :

- **Sequential** Each address is incremented by the **Sequential address increment** value that you specify. You also specify the **Start Address** from which the increments begin.
- **Random** Each address is generated randomly. (You set the number of random addresses on the **Loops** tab.)
- **Random Sequential** Each address is generated randomly, and then incremented sequentially. You specify the number of sequential increments in the **Number of sequential addresses** field.



The **Address Configuration** section contains the settings with which you configure the address mode that you chose in the **Address Mode** section. The following settings are available:

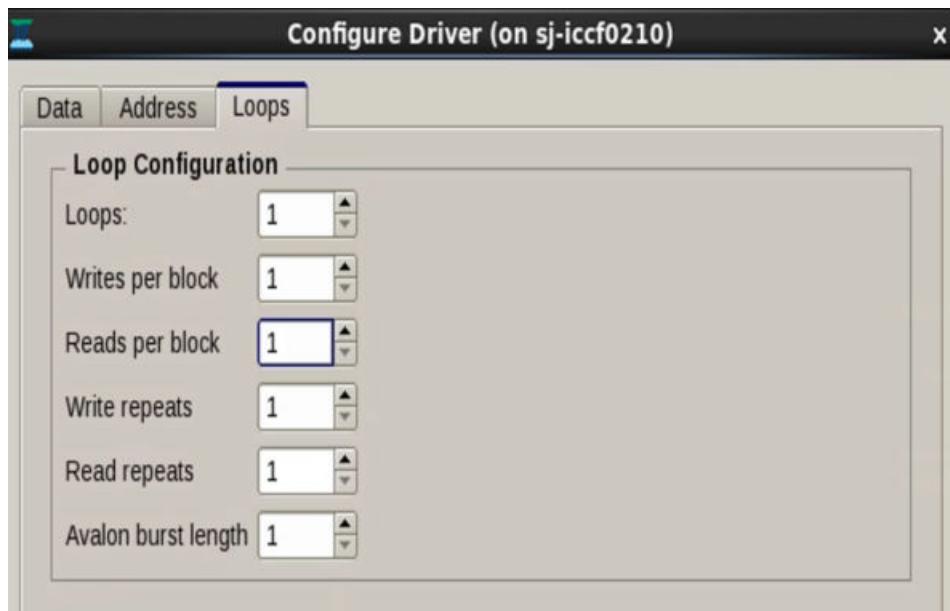
- **Start address** Specifies the starting address for **Sequential Address Mode**. The maximum address value that can be reached is 1FF\_FFFF. (The Traffic Generator 2.0 will accept higher values, but wraps back to 0 after the maximum value has been reached.) The **Start address** setting applies only to **Sequential Address Mode**.
- **Number of sequential addresses** Specifies the number of sequential addresses generated after the first random address generated. This setting applies only in **Random sequential** mode.
- **Sequential address increment** Specifies the size of increment between each address in the **Sequential** address mode and **Random sequential** address mode.
- **Return to start address** Specifies that the address value generated return back to the value entered in the **Start Address** field, after a block of transactions to the memory has completed. This setting applies only to **Sequential** address mode.
- **Address masking** Masking provides additional options for exploring certain specific address spaces in memory:
  - **Disabled** does not enable masking, and increments address based on the selected **Address Mode**.
  - **Fixed cycling** allows you to restrict the addressing to a specific row or a specific bank, which you can specify in the corresponding **Mask Value** field.

### Loops Tab

The **Loops** tab lets you order the transactions to the memory as desired. A unit size of transactions to the memory is defined as a *block*; a *block* includes a set of write transaction(s) immediately followed by a set of read transaction(s).



Figure 218. Loops Tab



The **Loops** tab provides the following configuration options:

- **Loops** Specifies the number of blocks of transactions to be sent to the memory. This option helps to extend the range of addresses that the controller can access. The address range is incremented as each loop is executed, unless you specify **Return to Start Address**, which causes each loop to begin from the same start address. The range of supported values for the **Loops** option is from 1 to 4095.
- **Writes per block** Specifies the size of the block (that is, the number of consecutive write operations that can be issued in a single block). The range of values for this option is as follows:
  - When address masking is disabled, the number of writes per block supported is 1 to 4094.
  - When address masking is enabled, the maximum number of writes issued inside a block is 255.
- **Reads per block** Specifies the number of consecutive read operations that can be issued in a single block, immediately following the consecutive writes issued. The number of reads per block should be identical to the number of writes per block, because data mismatches can occur when the two values are not identical. The range of values for this option is as follows:
  - When address masking is disabled, the number of reads per block supported is 1 to 4094.
  - When address masking is enabled, the maximum number of reads issued inside a block is 255.

- **Write repeats** Specifies the number of times each write command is issued in repetition to the same address. A maximum number of 255 repeat write transactions can be issued. The repeat writes are issued immediately after the first write command has been issued.
- **Read repeats** Specifies the number of times each read command is issued in repetition to the same address. A maximum number of 255 repeat read transactions can be issued. The repeat reads are issued immediately after the first read command has been issued.
- **Avalon burst length** Specifies the length of each Avalon burst. The value of this field should be less than the **Sequential address increment** specified on the **Address** tab. The number of write and read repeats default to 1 if the Avalon burst length is greater than 1.

#### 17.10.4. Applying the Traffic Generator 2.0

You can apply the Traffic Generator 2.0 to run stress tests, debug your hardware platform for signal integrity problems, and to emulate actual memory transactions. This topic presents some common applications where you can benefit by using the Traffic Generator 2.0.

##### Testing Signal Integrity with PRBS Data Pattern

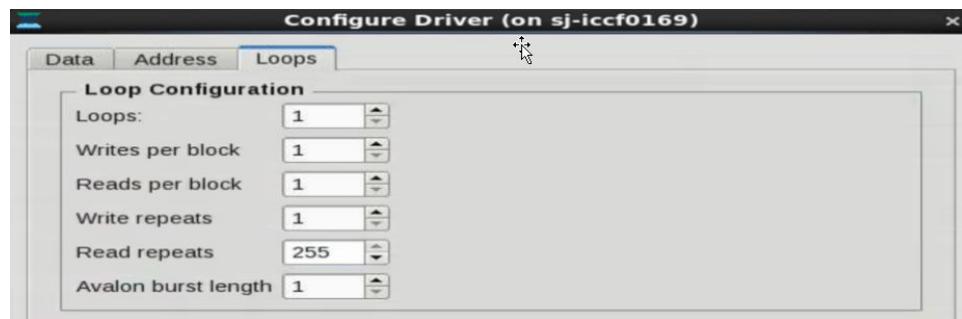
You can apply PRBS data to the data pins to help emulate an actual traffic pattern to the memory interface. The traffic generator uses a PRBS7 data pattern as the default traffic pattern on the data pins, and can support PRBS-15 and PRBS-31.

##### Debugging and Monitoring an Address for Reliable Data Capture

You can send a single write followed by multiple reads to a specific address to help debug and monitor a specific address for reliable data capture. You can do this with the following settings:

- Writes per block: 1
- Reads per block: 1
- Write repeats: 1
- Read repeats: 1 to 255

**Figure 219. Configuring the Loops Tab for a Single Write Followed by Multiple Reads**





If you specify a **Loops** value greater than 1, every block of write and multiple read transactions will follow the same pattern. If there is a specific address to which this transaction must be issued, you should specify that address in the **Start address** field on the **Address** tab, with the **Sequential** address mode selected.

### Accessing Large Sections of Memory

The maximum number of unique addresses that can be written to in one block is 4094. Using the maximum **Loops** value of 4095, the address range that can be supported in one test is equal to the number of loops multiplied by the number of writes per block. Further address expansion can be achieved by changing the **Start address** value appropriately and reissuing the tests.

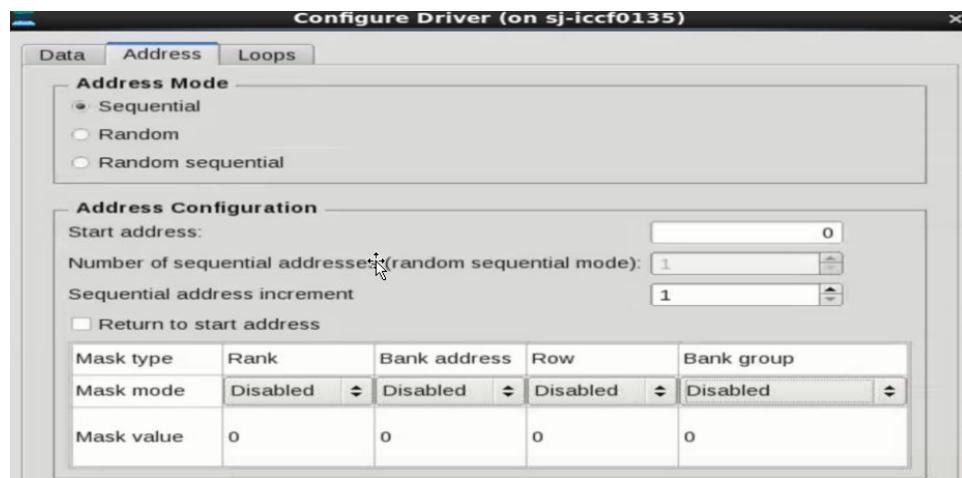
To continue addressing sections of the memory beyond the address range that can be specified in one set of toolkit configurations, you can incrementally access the next set of addresses in the memory by changing the **Start address** value.

For example, in a memory where row address width is 15, bank address width is 3 and column address width is 10, the total number of address locations to be accessed is:  $2 \text{ (row address width)} \times (\text{bank address width} \times 2 \text{ (column address width)})$ . The maximum number of address locations that can be accessed is limited by the width of the internal address bus, which is 25 bits wide.

For the example described above, you must set the following values on the **Address** tab:

- Select the **Sequential** address mode.
- Set the **Start address** to 0x00.
- Ensure that you do not select **Return to start address**.
- Ensure that you disable address masking for rank, row, bank, and bank group.

**Figure 220. Address Configuration to Access the First Set of Addresses**



You must also set the following values on the **Loops** tab:

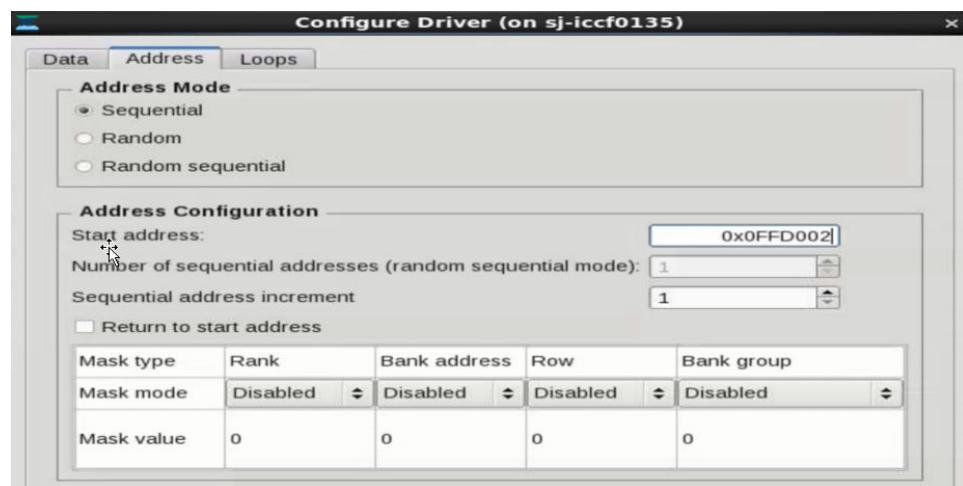
- Set **Loops** to the maximum value of 4095.
- Set **Writes per block** to the maximum value of 4094.
- Set **Reads per block** to the maximum value of 4094.
- Set **Write repeats** to 1.
- Set **Read repeats** to 1.

**Figure 221. Loop Configuration to Access the First Set of Addresses**



Each iteration can access a maximum of 4095 x 4094 locations (16,764,930 address locations i.e. Address ranging from 000\_0000'h to FF\_D001'h). To access the next 4095 x 4094 locations, the same settings as above must be repeated, except for the **Start address** value, which must be set to a hex value of 16,764,931 i.e. FF\_D002. The same process can be repeated to further access memory locations inside the memory. The maximum value supported is 25'h 1FF\_FFFF which is the equivalent of 33,554,432 locations inside the memory.

**Figure 222. Address Configuration to Access the Second Set of Addresses**



## 17.11. The Traffic Generator 2.0 Report

The traffic generator report provides information about the configuration of the Traffic Generator 2.0 and the result of the most recent run of traffic.



## Understanding the Traffic Generator 2.0 Report

The traffic generator report contains the following information:

- A **Pass flag** value of 1 indicates that the run completed with no errors.
- A **Fail flag** value of 1 indicates that the run encountered one or more errors.
- The **Failure Count** indicates the number of read transactions where data did not match the expected value.
- The **First failure** address indicates the address corresponding to the first data mismatch.
- The **Version** indicates the version number of the traffic generator.
- The **Number of data generators** indicates the number of data pins at the memory interface.
- The **Number of byte enable generators** indicates the number of byte enable and data mask pins at the memory interface.
- The **Rank Address**, **Bank address**, and **Bank group width** values indicate the number of bits in the Avalon address corresponding to each of those components.
- The **Data/Byte enable pattern length** indicates the number of bits in the fixed pattern used on each data/byte enable pin.
- The **PNF** (pass not fail) value indicates the persistent pass/fail status for each bit in the Avalon data. It is also presented on a per-memory-pin basis for each beat within a memory burst.
- **Fail Expected Data** is the data that was expected on the first failing transaction (if applicable).
- **Fail Read Data** is the data that was received on the first failing transaction (if applicable).

## 17.12. Example Tcl Script for Running the EMIF Debug Toolkit

If you want, you can run the EMIF Debug Toolkit using a Tcl script. The following example Tcl script is applicable to all device families.

The following example Tcl script opens a file, runs the debug toolkit, and writes the resulting calibration reports to a file.

You should adjust the variables in the script to match your design. You can then run the script using the command `quartus_sh -t example.tcl`.

```
# Modify the following variables for your project
set project "ed_synth.qpf"
# Index of the programming cable. Can be listed using "get_hardware_names"
set hardware_index 1
# Index of the device on the specified cable. Can be listed using
# "get_device_names"
set device_index 1
# SOF file containing the EMIF to debug
set sof "ed_synth.sof"
# Connection ID of the EMIF debug interface. Can be listed using
# "get_connections"
set connection_id 2
# Output file
set report "toolkit.rpt"

# The following code opens a project and writes its calibration reports to a
file.
```



```
project_open $project
load_package ::quartus::external_memif_toolkit
initialize_connections
set hardware_name [lindex [get_hardware_names] $hardware_index]
set device_name [lindex [get_device_names -hardware_name $hardware_name] $device_index]
link_project_to_device -device_name $device_name -hardware_name
$hardware_name -sof_file $sof
establish_connection -id $connection_id
create_connection_report -id $connection_id -report_type summary
create_connection_report -id $connection_id -report_type calib
write_connection_target_report -id $connection_id -file $report
```

## 17.13. Calibration Adjustment Delay Step Sizes for Arria 10 Devices

Refer to the following tables for information on delay step sizes for calibration adjustment.

### 17.13.1. Addressing

Each reconfigurable feature of the interface has an associated memory address; however, this address is placement dependent. If Altera PHYLite for Parallel Interfaces IP cores and the Arria 10 External Memory Interfaces IP cores share the same I/O column, you must track the addresses of the interface lanes and the pins. Addressing is done at the 32-bit word boundary, where avl\_address[1:0] are always 00.

#### Address Map

These points apply to the following table:

- id[3:0] refers to the **Interface ID** parameter.
- lane\_addr[7:0] refers to the address of a given lane in an interface. The Fitter sets this address value. You can query this in the Parameter Table Lookup Operation Sequence as described in *Address Lookup* section of the *Intel PHYLite for Parallel Interfaces IP Core User Guide*.
- pin[4:0] refers to the physical location of the pin in a lane. You can use the Fitter to automatically determine a pin location or you can manually set the pin location through .qsf assignment. Refer to the Parameter Table Lookup Operation Sequence as described in *Address Lookup* section of the *Intel PHYLite for Parallel Interfaces IP Core User Guide* for more information.

| Feature          | Avalon Address R/W   | Address CSR R  | Control        |       | Value   |
|------------------|--|--|----------------|-------|---|
|                  |  |  | Field          | Range |   |
| Pin Output Phase | {id[3:0],<br>3'h4,lan <sub>e</sub> _addr[7:<br>0],pin{4:0},8'D0} | {id[3:0],<br>3'h4,lan <sub>e</sub> _addr[7:<br>0],pin{4:0},<br>8'E8} | Phase<br>Value | 12..0 | Minimum Setting: Refer to <a href="#">Table 137</a> on page 503<br>Maximum Setting: Refer to <a href="#">Table 137</a> on page 503<br>Incremental Delay:<br>1/128th VCO clock period<br><b>continued...</b> |



| Feature   | Avalon Address R/W   | Address CSR R | Control   |        | Value  |
|---|--|---------------|---|--------|--|
|   |  |               | Field   | Range  |  |
| Pin PVT Compensated Input Delay                 | {id[3:0],<br>3'h4, lane_addr[7:<br>0],<br>4'hC, lgc_sel[1:0]<br>, pin_off[2:0],<br>4'h0}<br><ul style="list-style-type: none"><li>• lgc_sel[1:0] is:<ul style="list-style-type: none"><li>— 2'b01 for DQ [5:0]</li><li>— 2'b10 for DQ [11:6]</li></ul></li><li>• pin_off[2:0] :<ul style="list-style-type: none"><li>— 3'h0: DQ [0], DQ [6]</li><li>— 3'h1: DQ [1], DQ [7]</li><li>— 3'h2: DQ [2], DQ [8]</li><li>— 3'h3: DQ [3], DQ [9]</li><li>— 3'h4: DQ [4], DQ [10]</li><li>— 3'h5: DQ [5], DQ [11]</li></ul></li></ul> | Not supported | Delay Value   | 8..0   | Minimum Setting: 0<br>Maximum Setting: 511<br>VCO clock periods<br>Incremental Delay:<br>1/256th VCO clock period  |
|   |  |               | Reserved 1  | 11..9  | —  |
| Strobe PVT compensated input delay <sup>2</sup> | {id[3:0],<br>3'h4, lane_addr[7:<br>0],<br>4'hC, lgc_sel[1:0]<br>, 3'h6, 4'h0}<br><ul style="list-style-type: none"><li>• lgc_sel[1:0] = 2'b01</li></ul>  | Not supported | Enable  | 12     | 0 = Delay value is 0.<br>1 = Select delay value from Avalon register   |
|   |  |               | Reserved 1  | 31..13 | —  |
|   |  |               | Delay Value   | 9..0   | Minimum Setting: 0<br>Maximum Setting: 1023<br>VCO clock periods<br>Incremental Delay:<br>1/256th VCO clock period |
|   |  |               | Reserved 1  | 11..10 | —  |
|   | Enable   | 12            | 0 = Select delay value from CSR register. The CSR value is set through the <b>Capture Strobe Phase Shift</b> parameter during IP core instantiation.<br>1 = Select delay value from Avalon register |        |  |
|   |  |               | Reserved 1  | 31..13 | —  |

*continued...*



| Feature                          | Avalon Address R/W   | Address CSR R  | Control                  |        | Value   |
|----------------------------------|--|--|--------------------------|--------|---|
|                                  |  |  | Field                    | Range  |   |
| Strobe enable phase <sup>2</sup> | {id[3:0],<br>3'h4, lane_addr[7:<br>0],<br>4'hC, lgc_sel[1:0]<br>, 3'h7, 4'h0}<br>• lgc_sel[1:0] =<br>2'b01 | {id[3:0],<br>3'h4, lane_addr[7:<br>0], 4'hC, 9'h198} | Phase<br>Value           | 12..0  | Minimum Setting: Refer to <a href="#">Table 137</a> on page 503<br>Maximum Setting: Refer to <a href="#">Table 137</a> on page 503<br>Incremental Delay: 1/128th VCO clock period |
|                                  |  |  | Reserved<br><sub>1</sub> | 14..13 | —   |
|                                  |  |  | Enable                   | 15     | 0 = Select delay value from CSR register<br>1 = Select delay value from Avalon register   |
|                                  |  |  | Reserved<br><sub>1</sub> | 31..16 | —   |
| Strobe enable delay <sup>2</sup> | {id[3:0],<br>3'h4, lane_addr[7:<br>0], 4'hC, 9'h008}   | {id[3:0],<br>3'h4, lane_addr[7:<br>0], 4'hC, 9'h1A8} | Delay<br>Value           | 5..0   | Minimum Setting: 0 external clock cycles<br>Maximum Setting: 63 external memory clock cycles<br>Incremental Delay: 1 external memory clock cycle                                  |
|                                  |  |  | Reserved<br><sub>1</sub> | 14..6  | —   |
|                                  |  |  | Enable                   | 15     | 0 = Select delay value from CSR register<br>1 = Select delay value from Avalon register   |
|                                  |  |  | Reserved<br><sub>1</sub> | 31..16 | —   |
| Read valid delay <sup>2</sup>    | {id[3:0],<br>3'h4, lane_addr[7:<br>0], 4'hC, 9'h00C}   | {id[3:0],<br>3'h4, lane_addr[7:<br>0], 4'hC, 9'h1A4} | Delay<br>Value           | 6..0   | Minimum Setting: 0 external clock cycles<br>Maximum Setting: 127 external memory clock cycles<br>Incremental Delay: 1 external memory clock cycle                                 |
|                                  |  |  | Reserved<br><sub>1</sub> | 14..7  | —   |
|                                  |  |  | Enable                   | 15     | 0 = Select delay value from CSR register<br>1 = Select delay value from Avalon register   |
|                                  |  |  | Reserved<br><sub>1</sub> | 31..16 | —   |
| Internal VREF Code               | {id[3:0],<br>3'h4, lane_addr[7:<br>0], 4'hC, 9'h014}   | Not supported  | VREF<br>Code             | 5..0   | Refer to <i>Calibrated VREF Settings</i> in the <i>Intel® PHYLite for Parallel Interfaces IP Core User Guide</i> .  |

*continued...*



| Feature   | Avalon Address R/W | Address CSR R | Control       |                    | Value |
|---|--------------------|---------------|---------------|--------------------|-------|
|   |                    |               | Field         | Range              |       |
|   |                    |               | Reserved<br>1 | 31..6 <sup>9</sup> | —     |
| 1. Reserved bit ranges must be zero.<br>2. Modifying these values must be done on all lanes in a group. |                    |               |               |                    |       |

**Note:** For more information about performing various clocking and delay calculations, depending on the interface frequency and rate, refer to [PHYLite\\_delay\\_calculations.xlsx](#).

### 17.13.2. Output and Strobe Enable Minimum and Maximum Phase Settings

When dynamically reconfiguring the interpolator phase settings, the values must be kept within the ranges below to ensure proper operation of the circuitry.

**Table 137. Output and Strobe Enable Minimum and Maximum Phase Settings**

| VCO Multiplication Factor | Core Rate | Minimum Interpolator Phase |               |                                | Maximum Interpolator Phase |
|---------------------------|-----------|----------------------------|---------------|--------------------------------|----------------------------|
|                           |           | Output                     | Bidirectional | Bidirectional with OCT Enabled |                            |
| 1                         | Full      | 0x080                      | 0x100         | 0x100                          | 0xA80                      |
|                           | Half      | 0x080                      | 0x100         | 0x100                          | 0xBC0                      |
|                           | Quarter   | 0x080                      | 0x100         | 0x100                          | 0xA00                      |
| 2                         | Full      | 0x080                      | 0x100         | 0x180                          | 0x1400                     |
|                           | Half      | 0x080                      | 0x100         | 0x180                          | 0x1400                     |
|                           | Quarter   | 0x080                      | 0x100         | 0x180                          | 0x1400                     |
| 4                         | Full      | 0x080                      | 0x100         | 0x280                          | 0x1FFF                     |
|                           | Half      | 0x080                      | 0x100         | 0x280                          | 0x1FFF                     |
|                           | Quarter   | 0x080                      | 0x100         | 0x280                          | 0x1FFF                     |
| 8                         | Full      | 0x080                      | 0x100         | 0x480                          | 0x1FFF                     |
|                           | Half      | 0x080                      | 0x100         | 0x480                          | 0x1FFF                     |
|                           | Quarter   | 0x080                      | 0x100         | 0x480                          | 0x1FFF                     |

For more information about performing various clocking and delay calculations, depending on the interface frequency and rate, refer to [PHYLite\\_delay\\_calculations.xlsx](#).

### 17.14. Using the EMIF Debug Toolkit with Arria 10 HPS Interfaces

The External Memory Interface Debug Toolkit is not directly compatible with Arria 10 HPS interfaces.



To debug your Arria 10 HPS interface using the EMIF Debug Toolkit, you should create an identically parameterized, non-HPS version of your interface, and apply the EMIF Debug Toolkit to that interface. When you finish debugging this non-HPS interface, you can then apply any needed changes to your HPS interface, and continue your design development.

## 17.15. Document Revision History

| Date          | Version    | Changes   |
|---------------|------------|---|
| May 2017      | 2017.05.08 | <ul style="list-style-type: none"><li>Added <i>Using the EMIF Debug Toolkit with Arria 10 HPS Interfaces</i> topic.</li><li>Added <i>Calibration Adjustment Delay Step Sizes for Arria 10 Devices</i> topic.</li><li>Replaced <i>EMIF Configurable Traffic Generator 2.0</i> section with new <i>Traffic Generator 2.0</i> section.</li><li>Rebranded as Intel.</li></ul>   |
| October 2016  | 2016.10.31 | Maintenance release.  |
| May 2016      | 2016.05.02 | <ul style="list-style-type: none"><li>Added additional option to step 1 of <i>Establishing Communication to Connections</i>.</li><li>Added sentence to second bullet in <i>Eye Diagram</i>.</li><li>Expanded step 4 and added step 5, in <i>Determining Margin</i>.</li><li>Added <i>Configuring the Traffic Generator 2.0</i> and <i>The Traffic Generator 2.0 Report</i>.</li></ul>   |
| November 2015 | 2015.11.02 | <ul style="list-style-type: none"><li>Changed title of <i>Architecture</i> section to <i>User Interface</i>.</li><li>Added sentence to <i>Driver Margining</i> section stating that driver margining is not available if ECC is enabled.</li><li>Removed note that the memory map for Arria 10 On-Chip Debug would be available in a future release.</li><li>Created separate On-Chip Debug sections for UniPHY-based EMIF IP and Arria 10 EMIF IP.</li><li>Changed title of <i>Driver Margining (Arria 10 only)</i> section to <i>Driver Margining for Arria 10 EMIF IP</i>.</li><li>Changed title of <i>Read Setting and Apply Setting Commands (Arria 10 only)</i> to <i>Read Setting and Apply Setting Commands for Arria 10 EMIF IP</i>.</li><li>Added section <i>Example Tcl Script for Running the EMIF Debug Toolkit</i>.</li><li>Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i>.</li></ul> |
| May 2015      | 2015.05.04 | <ul style="list-style-type: none"><li>Added <i>Determining the Failing Calibration Stage for a Cyclone V or Arria V HPS SDRAM Controller</i>.</li><li>Changed occurrences of <i>On-Chip Debug Toolkit</i> to <i>On-Chip Debug Port</i>.</li><li>Added <i>Driver Margining (Arria 10 only)</i> and <i>Determining Margin</i>.</li><li>Added <i>Read Setting and Apply Setting Commands (Arria 10 only)</i> and <i>Reading or Applying Calibration Settings</i>.</li></ul>  |
| December 2014 | 2014.12.15 | <ul style="list-style-type: none"><li>Added paragraph to step 5 of <i>Generating IP With the Debug Port</i>.</li><li>Added mention of <code>seq_debug_clk</code> and <code>seq_debug_reset_in</code> to step 5 of <i>Generating IP With the Debug Port</i>.</li></ul>   |
| August 2014   | 2014.08.15 | Maintenance release.  |
| December 2013 | 2013.12.16 | Maintenance release.  |
| November 2012 | 2.2        | <ul style="list-style-type: none"><li>Changes to <i>Setup and Use</i> and <i>General Workflow</i> sections.</li><li>Added <i>EMIF On-Chip Debug Toolkit</i> section</li><li>Changed chapter number from 11 to 13.</li></ul>   |

*continued...*



| Date          | Version | Changes   |
|---------------|---------|---|
| August 2012   | 2.1     | Added table of debugging tips.  |
| June 2012     | 2.0     | <ul style="list-style-type: none"><li>Revised content for new UniPHY EMIF Toolkit.</li><li>Added Feedback icon.</li></ul> |
| November 2011 | 1.0     | Harvested 11.0 DDR2 and DDR3 SDRAM Controller with UniPHY EMIF Toolkit content.   |

## 18. Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers

The following topics describe the process of upgrading to UniPHY from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY Designs.

**NOTE:** Designs that do not use the AFI cannot be upgraded to UniPHY. If your design uses non-AFI IP cores, Intel recommends that you start a new design with the UniPHY IP core. In addition, Intel recommends that any new designs targeting Stratix III, Stratix IV, or Stratix V use the UniPHY datapath.

To upgrade your ALTMEMPHY-based DDR2 or DDR3 SDRAM High-Performance Controller II design to a DDR2 or DDR3 SDRAM controller with UniPHY IP core, you must complete the tasks listed below:

1. Generating Equivalent Design
2. Replacing the ALTMEMPHY Datapath with UniPHY Datapath
3. Resolving Port Name Differences
4. Creating OCT Signals
5. Running Pin Assignments Script
6. Removing Obsolete Files
7. Simulating your Design

The following topics describe these tasks in detail.

### Related Information

- [Generating Equivalent Design](#) on page 506
- [Replacing the ALTMEMPHY Datapath with UniPHY Datapath](#) on page 507
- [Resolving Port Name Differences](#) on page 507
- [Creating OCT Signals](#) on page 509
- [Running Pin Assignments Script](#) on page 509
- [Removing Obsolete Files](#) on page 509
- [Simulating your Design](#) on page 509
- [Creating OCT Signals](#) on page 509

### 18.1. Generating Equivalent Design

Create a new DDR2 or DDR3 SDRAM controller with UniPHY IP core, by following the steps in *Implementing and Parameterizing Memory IP* and apply the following guidelines:

---

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Empirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.

ISO  
9001:2015  
Registered



- Specify the same variation name as the ALTMEMPHY variation.
- Specify a directory different than the ALTMEMPHY design directory to prevent files from overwriting each other during generation.

To ease the migration process, ensure the UniPHY-based design you create is as similar as possible to the existing ALTMEMPHY-based design. In particular, you should ensure the following settings are the same in your UniPHY-based design:

- **PHY settings** tab
- FPGA speed grade
- PLL reference clock
- Memory clock frequency
- There is no need to change the default **Address and command clock phase settings**; however, if you have board skew effects in your ALTMEMPHY design, enter the difference between that clock phase and the default clock phase into the **Address and command clock phase settings**.
- **Memory Parameters** tab—all parameters must match.
- **Memory Timing** tab—all parameters must match.
- **Board settings** tab—all parameters must match.
- **Controller settings** tab—all parameters must match

**Note:** In ALTMEMPHY-based designs you can turn off dynamic OCT. However, all UniPHY-based designs use dynamic parallel OCT and you cannot turn it off.

#### Related Information

[Implementing and Parameterizing Memory IP](#)

## 18.2. Replacing the ALTMEMPHY Datapath with UniPHY Datapath

To replace the ALTMEMPHY datapath with the UniPHY datapath, follow these steps:

1. In the Quartus Prime software, open the Assignment Editor, on the Assignments menu click **Assignment Editor**.
2. Manually delete all of the assignments related to the external memory interface pins, except for the location assignments if you are preserving the pinout. By default, these pin names start with the `mem` prefix, though in your design they may have a different name.
3. Remove the old ALTMEMPHY **.qip** file from the project, as follows:
  - a. On the Assignments menu click **Settings**.
  - b. Specify the old **.qip**, and click **Remove**.

Your design now uses the UniPHY datapath.

## 18.3. Resolving Port Name Differences

Several port names in the ALTMEMPHY datapath are different than in the UniPHY datapath. The different names may cause compilation errors.



This topic describes the changes you must make in the RTL for the entity that instantiates the memory IP core. Each change applies to a specific port in the ALTMEMPHY datapath. Unconnected ports require no changes.

In some instances, multiple ports in ALTMEMPHY-based designs are mapped to a single port in UniPHY-based designs. If you use both ports in ALTMEMPHY-based designs, assign a temporary signal to the common port and connect it to the original wires. The following table shows the changes you must make.

**Table 138. Changes to ALTMEMPHY Port Names**

| ALTMEMPHY Port                       | Changes  |
|--------------------------------------|--|
| aux_full_rate_clk                    | The UniPHY-based design does not generate this signal. You can generate it if you require it.  |
| aux_scan_clk                         | The UniPHY-based design does not generate this signal. You can generate it if you require it.  |
| aux_scan_clk_reset_n                 | The UniPHY-based design does not generate this signal. You can generate it if you require it.  |
| dll_reference_clk                    | The UniPHY-based design does not generate this signal. You can generate it if you require it.  |
| dqs_delay_ctrl_export                | This signal is for DLL sharing between ALTMEMPHY instances and is not applicable for UniPHY-based designs.   |
| local_address                        | Rename to avl_addr.  |
| local_be                             | Rename to avl_be.  |
| local_burstbegin                     | Rename to avl_burstbegin.  |
| local_rdata                          | Rename to avl_rdata.   |
| local_rdata_valid                    | Rename to avl_rdata_valid.   |
| local_read_req                       | Rename to avl_read_req.  |
| local_ready                          | Rename to avl_ready.   |
| local_size                           | Rename to avl_size.  |
| local_wdata                          | Rename to avl_wdata.   |
| local_write_req                      | Rename to avl_write_req.   |
| mem_addr                             | Rename to mem_a.   |
| mem_clk                              | Rename to mem_ck.  |
| mem_clk_n                            | Rename to mem_ck_n.  |
| mem_dqsn                             | Rename to mem_dqs_n.   |
| oct_ctl_rs_value                     | Remove from design (see "Creating OCT Signals").   |
| oct_ctl_rt_value                     | Remove from design (see "Creating OCT Signals").   |
| phy_clk                              | Rename to afi_clk.   |
| reset_phy_clk_n                      | Rename to afi_reset_n.   |
| local_refresh_ack<br>reset_request_n | The controller no longer exposes these signals to the top-level design, so comment out these outputs. If you need it, bring the wire out from the high-performance II controller entity in <project_directory>/<variation name>.v. |



#### Related Information

[Creating OCT Signals](#) on page 509

### 18.4. Creating OCT Signals

In ALTMEMPHY-based designs, the Quartus Prime Fitter creates the alt\_oct block outside the IP core and connects it to the oct\_ctl\_rs\_value and oct\_ctl\_rt\_value signals.

In UniPHY-based designs, the OCT block is part of the IP core, so the design no longer requires these two ports. Instead, the UniPHY-based design requires two additional ports, oct\_rup and oct\_rdn (for Stratix III and Stratix IV devices), or oct\_rzqin (for Stratix V devices). You must create these ports in the instantiating entity as input pins and connect to the UniPHY instance. Then route these pins to the top-level design and connect to the OCT R<sub>UP</sub> and R<sub>DOWN</sub> resistors on the board.

For information on OCT control block sharing, refer to “The OCT Sharing Interface” in this volume.

### 18.5. Running Pin Assignments Script

Remap your design by running analysis and synthesis.

When analysis and synthesis completes, run the pin assignments Tcl script and then verify the new pin assignments in the Assignment Editor.

### 18.6. Removing Obsolete Files

After you upgrade the design, you may remove the unnecessary ALTMEMPHY design files from your project.

To identify these files, examine the original ALTMEMPHY-generated .qip file in any text editor.

### 18.7. Simulating your Design

You must use the UniPHY memory model to simulate your new design.

To use the UniPHY memory model, follow these steps:

1. Edit your instantiation of the UniPHY datapath to ensure the local\_init\_done, local\_cal\_success, local\_cal\_fail, soft\_reset\_n, oct\_rdn, oct\_rup, reset\_phy\_clk\_n, and phy\_clk signals are at the top-level entity so that an instantiating testbench can refer to those signals.
2. To use the UniPHY testbench and memory model, generate the example design when generating your IP instantiation.
3. Specify that your third-party simulator should use the UniPHY testbench and memory model instead of the ALTMEMPHY memory model, as follows:
  - a. On the Assignments menu, point to **Settings** and click the **Project Settings** window.
  - b. Select the **Simulation** tab, click **Test Benches**, click **Edit**, and replace the ALTMEMPHY testbench files with the following files:



- \<project directory>\<variation name>\_example\_design \simulation\verilog\submodules \altera\_avalon\_clock\_source.sv or \<project directory>\<variation name>\_example\_design\simulation\vhd1 \submodules\altera\_avalon\_clock\_source.vhd
  - \<project directory>\<variation name>\_example\_design \simulation\verilog\submodules \altera\_avalon\_reset\_source.sv or \<project directory>\<variation name>\_example\_design\simulation\vhd1 \submodules\altera\_avalon\_reset\_source.vhd
  - \<project directory>\<variation name>\_example\_design \simulation\verilog\<variation name>\_example\_sim.v or \uniphy\<variation name>\_example\_design\simulation\vhd1 \<variation name>\_example\_sim.vhd
  - \<project directory>\<variation name>\_example\_design \simulation\verilog\submodules\verbosity\_pkg.sv
  - \<project directory>\<variation name>\_example\_design \simulation\verilog\submodules \status\_checker\_no\_ifdef\_params.sv or \<project directory>\<variation name>\_example\_design\simulation\vhd1 \submodules\status\_checker\_no\_ifdef\_params.sv
  - \<project directory>\<variation name>\_example\_design \simulation\verilog\submodules \alt\_mem\_if\_common\_ddr\_mem\_model\_ddr3\_mem\_if\_dm\_pins\_en\_mem\_if\_dqsn\_en.sv or \<project directory>\<variation name>\_example\_design\simulation\vhd1\submodules \alt\_mem\_if\_common\_ddr\_mem\_model\_ddr3\_mem\_if\_dm\_pins\_en\_mem\_if\_dqsn\_en.sv
  - \<project directory>\<variation name>\_example\_design \simulation\verilog\submodules \alt\_mem\_if\_ddr3\_mem\_model\_top\_ddr3\_mem\_if\_dm\_pins\_en\_mem\_if\_dqsn\_en or \<project directory>\<variation name>\_example\_design\simulation\vhd1\submodules \alt\_mem\_if\_ddr3\_mem\_model\_top\_ddr3\_mem\_if\_dm\_pins\_en\_mem\_if\_dqsn\_en
4. Open the <variation name>\_example\_sim.v file and find the UniPHY-generated simulation example design module name below: <variation name>\_example\_sim\_e0.
  5. Change the module name above to the name of your top-level design module.
  6. Refer to the following table and update the listed port names of the example design in the UniPHY-generated <variation name>\_example\_sim.v file.



**Table 139. Example Design Port Names**

| Example Design Name | New Name  |
|---------------------|---|
| pll_ref_clk         | Rename to clock_source.   |
| mem_a               | Rename to mem_addr.   |
| mem_ck              | Rename to mem_clk.  |
| mem_ck_n            | Rename to mem_clk_n.  |
| mem_dqs_n           | Rename to mem_dqsn.   |
| drv_status_pass     | Rename to pnf.  |
| afi_clk             | Rename to phy_clk.  |
| afi_reset_n         | Rename to reset_phy_clk_n.  |
| drv_status_fail     | This signal is not available, so comment out this output.                       |
| afi_half_clk        | This signal is not exposed to the top-level design, so comment out this output. |

For more information about generating example simulation files, refer to Simulating Memory IP, in volume 2 of the *External Memory Interface Handbook*.

## 18.8. Document Revision History

| Date          | Version    | Changes  |
|---------------|------------|--|
| May 2017      | 2017.05.08 | Rebranded as Intel.  |
| October 2016  | 2016.10.31 | Maintenance release.   |
| May 2016      | 2016.05.02 | Maintenance release.   |
| November 2015 | 2015.11.02 | Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i> . |
| May 2015      | 2015.05.04 | Maintenance release.   |
| December 2014 | 2014.12.15 | Maintenance release.   |
| August 2014   | 2014.08.15 | Maintenance release.   |
| December 2013 | 2013.12.16 | Removed local_wdata_req from port names table.                   |
| November 2012 | 2.3        | Changed chapter number from 12 to 14.                            |
| June 2012     | 2.2        | Added Feedback icon.   |
| November 2011 | 2.1        | Revised <i>Simulating your Design</i> section.                   |