

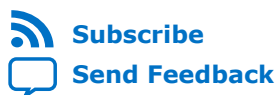


# Intel<sup>®</sup> Quartus<sup>®</sup> Prime Pro Edition Handbook Volume 1

## Design and Compilation

---

Updated for Intel<sup>®</sup> Quartus<sup>®</sup> Prime Design Suite: **17.1**



Subscribe

Send Feedback

**QPP5V1 | 2017.12.15**

Latest document on the web: [PDF](#) | [HTML](#)



## Contents

---

<b>1 Introduction to Intel® Quartus® Prime Pro Edition.....</b>	<b>16</b>
1.1 Should I Choose the Intel Quartus Prime Pro Edition Software?.....	17
1.2 Migrating to Intel Quartus Prime Pro Edition.....	18
1.2.1 Keep Pro Edition Project Files Separate.....	19
1.2.2 Upgrade Project Assignments and Constraints.....	19
1.2.3 Upgrade IP Cores and Platform Designer Systems.....	24
1.2.4 Upgrade Non-Compliant Design RTL.....	25
1.3 Document Revision History.....	30
<b>2 Managing Intel Quartus Prime Projects.....</b>	<b>32</b>
2.1 Understanding Intel Quartus Prime Projects.....	33
2.2 Viewing Basic Project Information.....	34
2.2.1 Viewing Project Reports.....	35
2.2.2 Viewing Project Messages.....	36
2.3 Using the Compilation Dashboard.....	37
2.4 Project Management Best Practices.....	38
2.5 Managing Project Settings.....	40
2.5.1 Optimizing Project Settings.....	42
2.6 Managing Logic Design Files.....	43
2.6.1 Including Design Libraries.....	44
2.7 Managing Timing Constraints.....	45
2.8 Introduction to Intel FPGA IP Cores.....	45
2.8.1 IP Catalog and Parameter Editor.....	46
2.8.2 Generating IP Cores (Intel Quartus Prime Pro Edition).....	50
2.8.3 Modifying an IP Variation.....	56
2.8.4 Upgrading IP Cores.....	56
2.8.5 Simulating Intel FPGA IP Cores.....	62
2.8.6 Synthesizing IP Cores in Other EDA Tools.....	71
2.8.7 Instantiating IP Cores in HDL.....	72
2.8.8 Support for the IEEE 1735 Encryption Standard.....	73
2.9 Integrating Other EDA Tools.....	73
2.10 Managing Team-based Projects.....	74
2.10.1 Preserving Compilation Results.....	74
2.10.2 Factors Affecting Compilation Results.....	75
2.10.3 Migrating Compilation Results Across Intel Quartus Prime Software Versions...	76
2.10.4 Archiving Projects.....	77
2.10.5 Using External Revision Control.....	78
2.10.6 Migrating Projects Across Operating Systems.....	79
2.11 Scripting API.....	81
2.11.1 Scripting Project Settings.....	81
2.11.2 Project Revision Commands.....	81
2.11.3 Project Archive Commands.....	82
2.11.4 Project Database Commands.....	83
2.11.5 Project Library Commands.....	84
2.12 Document Revision History.....	84
<b>3 Design Planning with the Intel Quartus Prime Software.....</b>	<b>87</b>
3.1 Design Planning with the Intel Quartus Prime Software.....	87



3.2	Creating Design Specifications.....	87
3.3	Selecting Intellectual Property Cores.....	88
3.4	Using Platform Designer and Standard Interfaces in System Design.....	88
3.5	Device Selection.....	89
3.5.1	Device Migration Planning.....	90
3.6	Development Kit Selection.....	90
3.6.1	Specifying a Development Kit for a New Project.....	90
3.6.2	Specifying a Development Kit for an Existing Project.....	91
3.6.3	Setting Pin Assignments.....	92
3.7	Planning for Device Programming or Configuration.....	92
3.8	Estimating Power.....	92
3.9	Selecting Third-Party EDA Tools.....	93
3.9.1	Synthesis Tool.....	94
3.9.2	Simulation Tool.....	94
3.9.3	Formal Verification Tools.....	94
3.10	Planning for On-Chip Debugging Tools.....	94
3.11	Design Practices and HDL Coding Styles.....	95
3.11.1	Design Recommendations.....	96
3.11.2	Recommended HDL Coding Styles.....	96
3.11.3	Managing Metastability.....	96
3.12	Running Fast Synthesis.....	97
3.13	Document Revision History.....	98
<b>4</b>	<b>Recommended HDL Coding Styles .....</b>	<b>100</b>
4.1	Using Provided HDL Templates.....	100
4.1.1	Inserting HDL Code from a Provided Template.....	100
4.2	Instantiating IP Cores in HDL.....	101
4.3	Inferring Multipliers and DSP Functions.....	102
4.3.1	Inferring Multipliers.....	102
4.3.2	Inferring Multiply-Accumulator and Multiply-Adder Functions.....	103
4.4	Inferring Memory Functions from HDL Code .....	104
4.4.1	Inferring RAM functions from HDL Code.....	105
4.4.2	Inferring ROM Functions from HDL Code.....	122
4.4.3	Inferring Shift Registers in HDL Code.....	124
4.5	Register and Latch Coding Guidelines.....	127
4.5.1	Register Power-Up Values.....	127
4.5.2	Secondary Register Control Signals Such as Clear and Clock Enable.....	129
4.5.3	Latches .....	130
4.6	General Coding Guidelines.....	133
4.6.1	Tri-State Signals .....	134
4.6.2	Clock Multiplexing.....	134
4.6.3	Adder Trees .....	136
4.6.4	State Machine HDL Guidelines.....	137
4.6.5	Multiplexer HDL Guidelines .....	143
4.6.6	Cyclic Redundancy Check Functions .....	145
4.6.7	Comparator HDL Guidelines.....	148
4.6.8	Counter HDL Guidelines.....	149
4.7	Designing with Low-Level Primitives.....	149
4.8	Document Revision History .....	150



- 5 Recommended Design Practices.....152**
  - 5.1 Following Synchronous FPGA Design Practices..... 152
    - 5.1.1 Implementing Synchronous Designs..... 152
    - 5.1.2 Asynchronous Design Hazards..... 153
  - 5.2 HDL Design Guidelines..... 154
    - 5.2.1 Considerations for the Intel Hyperflex FPGA Architecture..... 154
    - 5.2.2 Optimizing Combinational Logic.....155
    - 5.2.3 Optimizing Clocking Schemes..... 157
    - 5.2.4 Optimizing Physical Implementation and Timing Closure..... 163
    - 5.2.5 Optimizing Power Consumption.....166
    - 5.2.6 Managing Design Metastability..... 166
  - 5.3 Use Clock and Register-Control Architectural Features..... 166
    - 5.3.1 Use Global Reset Resources.....166
    - 5.3.2 Use Global Clock Network Resources.....176
    - 5.3.3 Use Clock Region Assignments to Optimize Clock Constraints..... 177
    - 5.3.4 Avoid Asynchronous Register Control Signals..... 179
  - 5.4 Implementing Embedded RAM..... 180
  - 5.5 Document Revision History..... 180
- 6 Design Compilation..... 183**
  - 6.1 Compilation Overview..... 184
    - 6.1.1 Compilation Flows..... 184
    - 6.1.2 Design Synthesis.....185
    - 6.1.3 Design Place and Route.....186
    - 6.1.4 Compilation Hierarchy.....187
    - 6.1.5 Reducing Compilation Time.....187
    - 6.1.6 Programming File Generation..... 188
  - 6.2 Running Full Compilation.....188
  - 6.3 Running Synthesis.....189
    - 6.3.1 Preserve Registers During Synthesis..... 190
    - 6.3.2 Enabling Timing-Driven Synthesis.....190
    - 6.3.3 Enabling Multi-Processor Compilation..... 191
    - 6.3.4 Synthesis Reports..... 191
  - 6.4 Running the Fitter..... 192
    - 6.4.1 Fitter Stage Commands.....193
    - 6.4.2 Incremental Optimization Flow..... 194
    - 6.4.3 Analyzing Fitter Snapshots..... 197
    - 6.4.4 Enabling Physical Synthesis Optimization.....203
    - 6.4.5 Viewing Fitter Reports.....204
  - 6.5 Running the Hyper-Aware Design Flow..... 207
    - 6.5.1 Step 1: Run Register Retiming..... 210
    - 6.5.2 Step 2: Review Retiming Results..... 211
    - 6.5.3 Step 3: Run Fast Forward Compile and Hyper-Retiming..... 213
    - 6.5.4 Step 4: Review Hyper-Retiming Results.....215
    - 6.5.5 Step 5: Implement Fast Forward Recommendations..... 218
  - 6.6 Running Rapid Recompile..... 221
  - 6.7 Generating Programming Files..... 222
  - 6.8 Synthesis Language Support.....223
    - 6.8.1 Verilog and SystemVerilog Synthesis Support.....223
    - 6.8.2 VHDL Synthesis Support..... 227



6.9 Synthesis Settings Reference.....	229
6.9.1 Optimization Modes.....	229
6.9.2 Prevent Register Retiming.....	229
6.9.3 Advanced Synthesis Settings.....	230
6.10 Fitter Settings Reference.....	236
6.11 Document Revision History.....	242
<b>7 Block-Based Design Flows.....</b>	<b>244</b>
7.1 Block-Based Design Examples.....	245
7.2 Design Partitioning.....	247
7.2.1 Planning Design Partitions.....	248
7.2.2 Creating and Modifying Design Partitions.....	250
7.2.3 Defining an Empty Partition.....	252
7.2.4 Top-Down, Bottom-Up, and Team-Based Design Methods.....	253
7.3 Incremental Block-Based Compilation.....	255
7.3.1 Define Empty Partitions to Reduce Compilation Time.....	256
7.4 Design Block Reuse.....	256
7.4.1 Reusing Core Partitions.....	257
7.4.2 Reusing Root Partitions.....	259
7.5 Debugging Block-Based Designs.....	262
7.5.1 Signal Tap with Core Partition Reuse.....	263
7.5.2 Signal Tap with Root Partition Reuse.....	265
7.6 Creating a Top-Level Project for a Team-Based Design.....	269
7.6.1 Preparing a Lower-Level Partition for Integration.....	270
7.7 Document Revision History.....	271
<b>8 Creating a Partial Reconfiguration Design.....</b>	<b>272</b>
8.1 Partial Reconfiguration Basic Concepts.....	273
8.2 Internal Host Partial Reconfiguration.....	275
8.3 External Host Partial Reconfiguration (Intel Arria 10 Designs Only).....	276
8.4 Partial Reconfiguration Design Flow.....	277
8.4.1 Identifying Partial Reconfiguration Resources.....	278
8.4.2 Defining PR Partitions.....	279
8.4.3 Defining Personas.....	281
8.4.4 Instantiating the Intel Arria 10 PR Controller IP.....	286
8.4.5 Instantiating the Intel Stratix 10 PR Controller IP.....	292
8.4.6 Promoting Global Signals in a PR Region.....	292
8.4.7 Partial Reconfiguration Process Sequence.....	293
8.4.8 Resetting the PR Region Registers.....	294
8.4.9 Floorplanning a Partial Reconfiguration Design.....	295
8.4.10 Creating Revisions for Personas.....	299
8.4.11 Compiling the Partial Reconfiguration Design.....	301
8.4.12 Timing Analysis with Partial Reconfiguration.....	306
8.4.13 External Host Configuration (Intel Arria 10 Designs Only).....	308
8.4.14 Programming File Generation.....	310
8.4.15 Partial Reconfiguration Design Debugging.....	317
8.4.16 Partial Reconfiguration Simulation and Verification.....	317
8.5 Partial Reconfiguration Design Recommendations.....	324
8.6 Partial Reconfiguration Design Considerations.....	325
8.7 Document Revision History.....	326



- 9 Creating a System with Platform Designer..... 327**
  - 9.1 Interface Support in Platform Designer..... 328
  - 9.2 Introduction to the Platform Designer IP Catalog..... 329
    - 9.2.1 Installing and Licensing IP Cores..... 329
    - 9.2.2 Adding IP Cores to IP Catalog..... 330
    - 9.2.3 General Settings for IP..... 331
    - 9.2.4 Set up the IP Index File (.ipx) to Search for IP Components ..... 332
    - 9.2.5 Integrate Third-Party IP Components into the Platform Designer IP Catalog.... 333
  - 9.3 Create a Platform Designer System..... 333
    - 9.3.1 Create/Open Project in Platform Designer..... 333
    - 9.3.2 Modify the Target Device..... 335
    - 9.3.3 Modify the IP Search Path..... 335
    - 9.3.4 Platform Designer System Design flow..... 336
    - 9.3.5 Add IP Components (IP Cores) to a Platform Designer System..... 337
    - 9.3.6 Specify Implementation Type for IP Components..... 338
    - 9.3.7 Connect IP Components in Your Platform Designer System..... 339
    - 9.3.8 Validate System Integrity..... 341
    - 9.3.9 Propagate System Information to IP Components..... 343
    - 9.3.10 View Your Platform Designer System..... 344
    - 9.3.11 Navigate Your Platform Designer System..... 350
    - 9.3.12 Specify IP Component Parameters..... 351
    - 9.3.13 Modify an Instantiated IP Component..... 354
    - 9.3.14 Save your System..... 355
    - 9.3.15 Archive your System..... 355
  - 9.4 Synchronize IP File References..... 356
  - 9.5 Upgrade Outdated IP Components in Platform Designer..... 356
  - 9.6 Create and Manage Hierarchical Platform Designer Systems..... 358
    - 9.6.1 Add a Subsystem to Your Platform Designer Design..... 358
    - 9.6.2 Drill into a Platform Designer Subsystem to Explore its Contents..... 359
    - 9.6.3 Edit a Platform Designer Subsystem..... 360
    - 9.6.4 Change the Hierarchy Level of a Platform Designer Component..... 361
    - 9.6.5 Save New Platform Designer Subsystem..... 361
  - 9.7 Specify Signal and Interface Boundary Requirements..... 361
    - 9.7.1 Match the Exported Interface with Interface Requirements..... 362
    - 9.7.2 Edit the Name of Exported Interfaces and Signals..... 363
  - 9.8 Run System Scripts..... 364
  - 9.9 View and Filter Clock and Reset Domains in Your Platform Designer System..... 365
    - 9.9.1 View Clock Domains in Your Platform Designer System..... 366
    - 9.9.2 View Reset Domains in Your Platform Designer System..... 367
    - 9.9.3 Filter Platform Designer Clock and Reset Domains in the System Contents Tab 368
    - 9.9.4 View Avalon Memory Mapped Domains in Your Platform Designer System..... 369
  - 9.10 Specify Platform Designer Interconnect Requirements..... 371
  - 9.11 Manage Platform Designer System Security..... 373
    - 9.11.1 Configure Platform Designer Security Settings Between Interfaces..... 374
    - 9.11.2 Specify a Default Slave in a Platform Designer System..... 374
    - 9.11.3 Access Undefined Memory Regions..... 375
  - 9.12 Integrating a Platform Designer System with a Intel Quartus Prime Project..... 376
  - 9.13 Manage IP Settings in the Intel Quartus Prime Software..... 376
    - 9.13.1 Opening Platform Designer with Additional Memory..... 377
  - 9.14 Generate a Platform Designer System..... 377



9.14.1	Set the Generation ID.....	378
9.14.2	Generate Files for Synthesis and Simulation.....	378
9.14.3	Generate Files for a Testbench Platform Designer System.....	382
9.14.4	Platform Designer Simulation Scripts.....	385
9.14.5	Simulating Software Running on a Nios II Processor.....	387
9.14.6	Add Assertion Monitors for Simulation.....	388
9.14.7	CMSIS Support for the HPS IP Component.....	389
9.14.8	Generate Header Files.....	389
9.14.9	Incrementally Generate the System.....	390
9.15	Explore and Manage Platform Designer Interconnect.....	391
9.15.1	Manually Controlling Pipelining in the Platform Designer Interconnect.....	392
9.16	Implement Performance Monitoring.....	394
9.17	Platform Designer 64-Bit Addressing Support.....	394
9.17.1	Support for Avalon-MM Non-Power of Two Data Widths.....	394
9.18	Platform Designer System Example Designs.....	395
9.19	Platform Designer Command-Line Utilities.....	395
9.19.1	Run the Platform Designer Editor with qsys-edit.....	395
9.19.2	Scripting IP Core Generation.....	396
9.19.3	Display Available IP Components with ip-catalog.....	398
9.19.4	Create an .ipx File with ip-make-ipx.....	398
9.19.5	Generate Simulation Scripts.....	399
9.19.6	Generate a Platform Designer System with qsys-script.....	400
9.19.7	Platform Designer Scripting Command Reference.....	401
9.19.8	Platform Designer Scripting Property Reference.....	574
9.19.9	Parameterizing an Instantiated IP Core after save_system Command.....	601
9.19.10	Validate the Generic Components in a System with qsys-validate.....	602
9.19.11	Archive a Platform Designer System with qsys-archive.....	602
9.19.12	Generate an IP Component or Platform Designer System with quartus_ipgenerate.....	603
9.19.13	Generate an IP Variation File with ip-deploy.....	605
9.20	Document Revision History.....	606
<b>10</b>	<b>Creating Platform Designer Components.....</b>	<b>608</b>
10.1	Platform Designer Components.....	608
10.1.1	Interface Support in Platform Designer.....	608
10.1.2	Component Structure.....	609
10.1.3	Component File Organization.....	610
10.1.4	Component Versions.....	610
10.2	Design Phases of an IP Component.....	611
10.3	Create IP Components in the Platform Designer Component Editor.....	612
10.3.1	Save an IP Component and Create the _hw.tcl File.....	614
10.3.2	Edit an IP Component with the Platform Designer Component Editor.....	614
10.4	Specify IP Component Type Information.....	614
10.5	Create an HDL File in the Platform Designer Component Editor.....	617
10.6	Create an HDL File Using a Template in the Platform Designer Component Editor.....	617
10.7	Specify Synthesis and Simulation Files in the Platform Designer Component Editor.....	618
10.7.1	Specify HDL Files for Synthesis in the Platform Designer Component Editor...	619
10.7.2	Analyze Synthesis Files in the Platform Designer Component Editor.....	620
10.7.3	Name HDL Signals for Automatic Interface and Type Recognition in the Platform Designer Component Editor.....	621
10.7.4	Specify Files for Simulation in the Component Editor.....	622



- 10.7.5 Include an Internal Register Map Description in the .svd for Slave Interfaces Connected to an HPS Component..... 623
- 10.8 Add Signals and Interfaces in the Platform Designer Component Editor.....624
- 10.9 Specify Parameters in the Platform Designer Component Editor..... 625
  - 10.9.1 Valid Ranges for Parameters in the \_hw.tcl File..... 628
  - 10.9.2 Types of Platform Designer Parameters.....628
  - 10.9.3 Declare Parameters with Custom \_hw.tcl Commands..... 630
  - 10.9.4 Validate Parameter Values with a Validation Callback..... 632
- 10.10 Declaring SystemVerilog Interfaces in \_hw.tcl.....632
- 10.11 User Alterable HDL Parameters in \_hw.tcl.....634
- 10.12 Control Interfaces Dynamically with an Elaboration Callback..... 635
- 10.13 Control File Generation Dynamically with Parameters and a Fileset Callback.....636
- 10.14 Create a Composed Component or Subsystem.....638
- 10.15 Add Component Instances to a Static or Generated Component..... 640
  - 10.15.1 Static Components.....640
  - 10.15.2 Generated Components..... 641
  - 10.15.3 Design Guidelines for Adding Component Instances.....644
- 10.16 Adding a Generic Component to the Platform Designer System..... 644
  - 10.16.1 Creating Custom Interfaces in a Generic Component..... 646
  - 10.16.2 Instantiating RTL in a System as a Generic Component ..... 649
  - 10.16.3 Implementing Generic Components Using High Level Synthesis Files..... 650
  - 10.16.4 Creating System Template for a Generic Component.....655
  - 10.16.5 Exporting a Generic Component.....657
- 10.17 Document Revision History..... 657
- 11 Platform Designer Interconnect..... 659**
  - 11.1 Memory-Mapped Interfaces..... 659
    - 11.1.1 Platform Designer Packet Format..... 661
    - 11.1.2 Interconnect Domains..... 664
    - 11.1.3 Master Network Interfaces..... 666
    - 11.1.4 Slave Network Interfaces.....669
    - 11.1.5 Arbitration..... 671
    - 11.1.6 Memory-Mapped Arbiter..... 675
    - 11.1.7 Datapath Multiplexing Logic..... 677
    - 11.1.8 Width Adaptation.....677
    - 11.1.9 Burst Adapter..... 679
    - 11.1.10 Read and Write Responses..... 681
    - 11.1.11 Platform Designer Address Decoding..... 682
  - 11.2 Avalon Streaming Interfaces..... 683
    - 11.2.1 Avalon-ST Adapters..... 685
  - 11.3 Interrupt Interfaces..... 693
    - 11.3.1 Individual Requests IRQ Scheme..... 693
    - 11.3.2 Assigning IRQs in Platform Designer..... 694
  - 11.4 Clock Interfaces..... 696
    - 11.4.1 (High Speed Serial Interface) HSSI Clock Interfaces.....697
  - 11.5 Reset Interfaces..... 702
    - 11.5.1 Single Global Reset Signal Implemented by Platform Designer..... 703
    - 11.5.2 Reset Controller..... 703
    - 11.5.3 Reset Bridge..... 703
    - 11.5.4 Reset Sequencer..... 704
  - 11.6 Conduits..... 715





11.7 Interconnect Pipelining.....	715
11.7.1 Manually Controlling Pipelining in the Platform Designer Interconnect.....	717
11.8 Error Correction Coding (ECC) in Platform Designer Interconnect.....	718
11.9 AMBA 3 AXI Protocol Specification Support (version 1.0).....	718
11.9.1 Channels.....	719
11.9.2 Cache Support.....	719
11.9.3 Security Support.....	720
11.9.4 Atomic Accesses.....	720
11.9.5 Response Signaling.....	720
11.9.6 Ordering Model.....	720
11.9.7 Data Buses.....	721
11.9.8 Unaligned Address Commands.....	721
11.9.9 Avalon and AXI Transaction Support.....	721
11.10 AMBA 3 APB Protocol Specification Support (version 1.0).....	722
11.10.1 Bridges.....	722
11.10.2 Burst Adaptation.....	722
11.10.3 Width Adaptation.....	723
11.10.4 Error Response.....	723
11.11 AMBA 4 AXI Memory-Mapped Interface Support (version 2.0).....	723
11.11.1 Burst Support.....	723
11.11.2 QoS.....	723
11.11.3 Regions.....	723
11.11.4 Write Response Dependency.....	724
11.11.5 AWCACHE and ARCACHE.....	724
11.11.6 Width Adaptation and Data Packing in Platform Designer.....	724
11.11.7 Ordering Model.....	724
11.11.8 Read and Write Allocate.....	725
11.11.9 Locked Transactions.....	725
11.11.10 Memory Types.....	725
11.11.11 Mismatched Attributes.....	725
11.11.12 Signals.....	725
11.12 AMBA 4 AXI Streaming Interface Support (version 1.0).....	725
11.12.1 Connection Points.....	725
11.12.2 Adaptation.....	726
11.13 AMBA 4 AXI-Lite Protocol Specification Support (version 2.0).....	726
11.13.1 AMBA 4 AXI-Lite Signals.....	727
11.13.2 AMBA 4 AXI-Lite Bus Width.....	727
11.13.3 AMBA 4 AXI-Lite Outstanding Transactions.....	727
11.13.4 AMBA 4 AXI-Lite IDs.....	727
11.13.5 Connections Between AMBA 3 AXI,AMBA 4 AXI and AMBA 4 AXI-Lite.....	727
11.13.6 AMBA 4 AXI-Lite Response Merging.....	728
11.14 Port Roles (Interface Signal Types).....	728
11.14.1 AXI Master Interface Signal Types.....	728
11.14.2 AXI Slave Interface Signal Types.....	729
11.14.3 AMBA 4 AXI Master Interface Signal Types.....	730
11.14.4 AMBA 4 AXI Slave Interface Signal Types.....	732
11.14.5 AMBA 4 AXI-Stream Master and Slave Interface Signal Types.....	733
11.14.6 APB Interface Signal Types.....	734
11.14.7 Avalon Memory-Mapped Interface Signal Roles.....	734
11.14.8 Avalon Streaming Interface Signal Roles .....	737
11.14.9 Avalon Clock Source Signal Roles .....	738



- 11.14.10 Avalon Clock Sink Signal Roles ..... 738
- 11.14.11 Avalon Conduit Signal Roles ..... 739
- 11.14.12 Avalon Tristate Conduit Signal Roles ..... 739
- 11.14.13 Avalon Tri-State Slave Interface Signal Types..... 740
- 11.14.14 Avalon Interrupt Sender Signal Roles ..... 741
- 11.14.15 Avalon Interrupt Receiver Signal Roles ..... 741
- 11.15 Document Revision History..... 741
- 12 Optimizing Platform Designer System Performance..... 743**
  - 12.1 Designing with Avalon and AXI Interfaces..... 743
    - 12.1.1 Designing Streaming Components..... 744
    - 12.1.2 Designing Memory-Mapped Components..... 744
  - 12.2 Using Hierarchy in Systems..... 745
  - 12.3 Using Concurrency in Memory-Mapped Systems..... 748
    - 12.3.1 Implementing Concurrency With Multiple Masters..... 749
    - 12.3.2 Implementing Concurrency With Multiple Slaves..... 750
    - 12.3.3 Implementing Concurrency with DMA Engines..... 752
  - 12.4 Inserting Pipeline Stages to Increase System Frequency ..... 753
  - 12.5 Using Bridges..... 753
    - 12.5.1 Using Bridges to Increase System Frequency..... 754
    - 12.5.2 Using Bridges to Minimize Design Logic..... 757
    - 12.5.3 Using Bridges to Minimize Adapter Logic..... 759
    - 12.5.4 Considering the Effects of Using Bridges..... 760
  - 12.6 Increasing Transfer Throughput..... 766
    - 12.6.1 Using Pipelined Transfers..... 767
    - 12.6.2 Arbitration Shares and Bursts..... 768
  - 12.7 Reducing Logic Utilization..... 772
    - 12.7.1 Minimizing Interconnect Logic to Reduce Logic Utilization..... 772
    - 12.7.2 Minimizing Arbitration Logic by Consolidating Multiple Interfaces..... 773
    - 12.7.3 Reducing Logic Utilization With Multiple Clock Domains..... 775
    - 12.7.4 Duration of Transfers Crossing Clock Domains ..... 777
  - 12.8 Reducing Power Consumption..... 778
    - 12.8.1 Reducing Power Consumption With Multiple Clock Domains..... 778
    - 12.8.2 Reducing Power Consumption by Minimizing Toggle Rates..... 781
    - 12.8.3 Reducing Power Consumption by Disabling Logic..... 782
  - 12.9 Reset Polarity and Synchronization in Platform Designer..... 783
  - 12.10 Optimizing Platform Designer System Performance Design Examples..... 786
    - 12.10.1 Avalon Pipelined Read Master Example..... 786
    - 12.10.2 Multiplexer Examples..... 788
  - 12.11 Document Revision History..... 789
- 13 Component Interface Tcl Reference..... 791**
  - 13.1 Platform Designer \_hw.tcl Command Reference..... 791
    - 13.1.1 Interfaces and Ports..... 792
    - 13.1.2 Parameters..... 810
    - 13.1.3 Display Items..... 821
    - 13.1.4 Module Definition..... 828
    - 13.1.5 Composition..... 840
    - 13.1.6 Fileset Generation..... 860
    - 13.1.7 Miscellaneous..... 871
    - 13.1.8 SystemVerilog Interface Commands..... 877



13.2 Platform Designer _hw.tcl Property Reference.....	883
13.2.1 Script Language Properties.....	884
13.2.2 Interface Properties.....	885
13.2.3 SystemVerilog Interface Properties.....	885
13.2.4 Instance Properties.....	886
13.2.5 Parameter Properties.....	887
13.2.6 Parameter Type Properties.....	889
13.2.7 Parameter Status Properties.....	890
13.2.8 Port Properties.....	891
13.2.9 Direction Properties.....	893
13.2.10 Display Item Properties.....	894
13.2.11 Display Item Kind Properties.....	895
13.2.12 Display Hint Properties.....	896
13.2.13 Module Properties.....	897
13.2.14 Fileset Properties.....	899
13.2.15 Fileset Kind Properties.....	900
13.2.16 Callback Properties.....	901
13.2.17 File Attribute Properties.....	902
13.2.18 File Kind Properties.....	903
13.2.19 File Source Properties.....	904
13.2.20 Simulator Properties.....	905
13.2.21 Port VHDL Type Properties.....	906
13.2.22 System Info Type Properties.....	907
13.2.23 Design Environment Type Properties.....	909
13.2.24 Units Properties.....	910
13.2.25 Operating System Properties.....	911
13.2.26 Quartus.ini Type Properties.....	912
13.3 Document Revision History.....	913
<b>14 Platform Designer System Design Components.....</b>	<b>914</b>
14.1 Bridges.....	914
14.1.1 Clock Bridge.....	915
14.1.2 Avalon-MM Clock Crossing Bridge.....	916
14.1.3 Avalon-MM Pipeline Bridge.....	918
14.1.4 Avalon-MM Unaligned Burst Expansion Bridge.....	919
14.1.5 Bridges Between Avalon and AXI Interfaces.....	922
14.1.6 AXI Bridge.....	923
14.1.7 AXI Timeout Bridge.....	928
14.1.8 Address Span Extender.....	932
14.2 Error Response Slave.....	937
14.2.1 Error Response Slave Parameters.....	938
14.2.2 Error Response Slave CSR Registers.....	939
14.2.3 Designating a Default Slave in the System Contents Tab.....	942
14.3 Tri-State Components.....	943
14.3.1 Generic Tri-State Controller.....	945
14.3.2 Tri-State Conduit Pin Sharer.....	945
14.3.3 Tri-State Conduit Bridge.....	946
14.4 Test Pattern Generator and Checker Cores.....	946
14.4.1 Test Pattern Generator.....	947
14.4.2 Test Pattern Checker.....	949



- 14.4.3 Software Programming Model for the Test Pattern Generator and Checker Cores..... 950
- 14.4.4 Test Pattern Generator API.....954
- 14.4.5 Test Pattern Checker API..... 959
- 14.5 Avalon-ST Splitter Core..... 966
  - 14.5.1 Splitter Core Backpressure.....966
  - 14.5.2 Splitter Core Interfaces..... 967
  - 14.5.3 Splitter Core Parameters..... 967
- 14.6 Avalon-ST Delay Core..... 968
  - 14.6.1 Delay Core Reset Signal..... 968
  - 14.6.2 Delay Core Interfaces..... 968
  - 14.6.3 Delay Core Parameters.....969
- 14.7 Avalon-ST Round Robin Scheduler..... 970
  - 14.7.1 Almost-Full Status Interface (Round Robin Scheduler)..... 970
  - 14.7.2 Request Interface (Round Robin Scheduler)..... 970
  - 14.7.3 Round Robin Scheduler Operation..... 970
  - 14.7.4 Round Robin Scheduler Parameters.....971
- 14.8 Avalon Packets to Transactions Converter..... 972
  - 14.8.1 Packets to Transactions Converter Interfaces.....972
  - 14.8.2 Packets to Transactions Converter Operation..... 972
- 14.9 Avalon-ST Streaming Pipeline Stage.....974
- 14.10 Streaming Channel Multiplexer and Demultiplexer Cores..... 975
  - 14.10.1 Software Programming Model For the Multiplexer and Demultiplexer Components..... 976
  - 14.10.2 Avalon-ST Multiplexer.....976
  - 14.10.3 Avalon-ST Demultiplexer..... 978
- 14.11 Single-Clock and Dual-Clock FIFO Cores..... 979
  - 14.11.1 Interfaces Implemented in FIFO Cores.....980
  - 14.11.2 FIFO Operating Modes.....981
  - 14.11.3 Fill Level of the FIFO Buffer.....982
  - 14.11.4 Almost-Full and Almost-Empty Thresholds to Prevent Overflow and Underflow..... 982
  - 14.11.5 Single-Clock and Dual-Clock FIFO Core Parameters..... 982
  - 14.11.6 Avalon-ST Single-Clock FIFO Registers.....983
- 14.12 Document Revision History..... 984
- 15 Managing Metastability with the Intel Quartus Prime Software..... 986**
  - 15.1 Metastability Analysis in the Intel Quartus Prime Software..... 987
    - 15.1.1 Synchronization Register Chains.....987
    - 15.1.2 Identify Synchronizers for Metastability Analysis..... 988
    - 15.1.3 How Timing Constraints Affect Synchronizer Identification and Metastability Analysis.....988
  - 15.2 Metastability and MTBF Reporting..... 989
    - 15.2.1 Metastability Reports.....990
    - 15.2.2 Synchronizer Data Toggle Rate in MTBF Calculation.....992
  - 15.3 MTBF Optimization..... 992
    - 15.3.1 Synchronization Register Chain Length.....993
  - 15.4 Reducing Metastability Effects.....994
    - 15.4.1 Apply Complete System-Centric Timing Constraints for the Timing Analyzer.. 994
    - 15.4.2 Force the Identification of Synchronization Registers..... 994
    - 15.4.3 Set the Synchronizer Data Toggle Rate..... 995



15.4.4 Optimize Metastability During Fitting.....	995
15.4.5 Increase the Length of Synchronizers to Protect and Optimize.....	995
15.4.6 Increase the Number of Stages Used in Synchronizers.....	995
15.4.7 Select a Faster Speed Grade Device.....	996
15.5 Scripting Support.....	996
15.5.1 Identifying Synchronizers for Metastability Analysis.....	996
15.5.2 Synchronizer Data Toggle Rate in MTBF Calculation.....	997
15.5.3 report_metastability and Tcl Command.....	997
15.5.4 MTBF Optimization.....	997
15.5.5 Synchronization Register Chain Length.....	998
15.6 Managing Metastability.....	998
15.7 Document Revision History.....	998
<b>16 Mitigating Single Event Upset.....</b>	<b>1000</b>
16.1 Failure Rates.....	1001
16.2 Mitigating SEU Effects in Embedded User RAM.....	1001
16.2.1 Configuring RAM to Enable ECC.....	1002
16.3 Mitigating SEU Effects in Configuration RAM (Intel Arria 10 and Intel Cyclone 10 GX devices).....	1003
16.4 Mitigating SEU Effects in Configuration RAM (Intel Stratix 10 devices).....	1004
16.4.1 Error Message Register.....	1004
16.4.2 SEU_ERROR Pin Behavior.....	1005
16.5 Internal Scrubbing.....	1005
16.6 SEU Recovery.....	1006
16.6.1 Planning for SEU Recovery.....	1006
16.6.2 Designating the Sensitivity of the Design Hierarchy .....	1007
16.6.3 Advanced SEU Detection IP Core.....	1009
16.7 Intel Quartus Prime Software SEU FIT Reports.....	1010
16.7.1 SEU FIT Parameters Report.....	1010
16.7.2 Projected SEU FIT by Component Usage Report.....	1011
16.7.3 Enabling the Projected SEU FIT by Component Usage Report.....	1014
16.8 Triple-Module Redundancy.....	1014
16.9 Evaluating a System's Response to Functional Upsets.....	1014
16.10 CRAM Error Detection Settings Reference.....	1015
16.11 Document Revision History.....	1016
<b>17 Optimizing the Design Netlist.....</b>	<b>1018</b>
17.1 When to Use the Netlist Viewers: Analyzing Design Problems .....	1018
17.2 Intel Quartus Prime Design Flow with the Netlist Viewers.....	1019
17.3 RTL Viewer Overview.....	1020
17.4 Technology Map Viewer Overview.....	1021
17.5 Introduction to the User Interface.....	1022
17.5.1 Netlist Navigator Pane.....	1025
17.5.2 Properties Pane.....	1025
17.5.3 Netlist Viewers Find Pane.....	1027
17.6 Schematic View.....	1027
17.6.1 Display Schematics in Multiple Tabbed View.....	1027
17.6.2 Schematic Symbols.....	1028
17.6.3 Select Items in the Schematic View.....	1031
17.6.4 Shortcut Menu Commands in the Schematic View.....	1031
17.6.5 Filtering in the Schematic View.....	1031
17.6.6 View Contents of Nodes in the Schematic View.....	1032



- 17.6.7 Moving Nodes in the Schematic View..... 1034
- 17.6.8 View LUT Representations in the Technology Map Viewer..... 1034
- 17.6.9 Zoom Controls..... 1034
- 17.6.10 Navigating with the Bird's Eye View..... 1035
- 17.6.11 Partition the Schematic into Pages..... 1035
- 17.6.12 Follow Nets Across Schematic Pages..... 1036
- 17.7 Cross-Probing to a Source Design File and Other Intel Quartus Prime Windows..... 1036
- 17.8 Cross-Probing to the Netlist Viewers from Other Intel Quartus Prime Windows..... 1037
- 17.9 Viewing a Timing Path..... 1037
- 17.10 Document Revision History..... 1038
- 18 Mentor Graphics Precision Synthesis Support..... 1040**
  - 18.1 About Precision RTL Synthesis Support..... 1040
  - 18.2 Design Flow..... 1040
    - 18.2.1 Timing Optimization..... 1042
  - 18.3 Intel Device Family Support..... 1042
  - 18.4 Precision Synthesis Generated Files..... 1042
  - 18.5 Creating and Compiling a Project in the Precision Synthesis Software..... 1043
  - 18.6 Mapping the Precision Synthesis Design..... 1043
    - 18.6.1 Setting Timing Constraints..... 1044
    - 18.6.2 Setting Mapping Constraints..... 1044
    - 18.6.3 Assigning Pin Numbers and I/O Settings..... 1044
    - 18.6.4 Assigning I/O Registers..... 1045
    - 18.6.5 Disabling I/O Pad Insertion..... 1045
    - 18.6.6 Controlling Fan-Out on Data Nets..... 1046
  - 18.7 Synthesizing the Design and Evaluating the Results..... 1046
    - 18.7.1 Obtaining Accurate Logic Utilization and Timing Analysis Reports..... 1047
  - 18.8 Guidelines for Intel FPGA IP Cores and Architecture-Specific Features..... 1047
    - 18.8.1 Instantiating IP Cores With IP Catalog-Generated Verilog HDL Files..... 1047
    - 18.8.2 Instantiating IP Cores With IP Catalog-Generated VHDL Files..... 1048
    - 18.8.3 Instantiating Intellectual Property With the IP Catalog and Parameter Editor..... 1048
    - 18.8.4 Instantiating Black Box IP Functions With Generated Verilog HDL Files..... 1049
    - 18.8.5 Instantiating Black Box IP Functions With Generated VHDL Files..... 1049
    - 18.8.6 Inferring Intel FPGA IP Cores from HDL Code..... 1050
  - 18.9 Document Revision History..... 1055
- 19 Synopsys Synplify Support..... 1056**
  - 19.1 About Synplify Support..... 1056
  - 19.2 Design Flow..... 1056
  - 19.3 Hardware Description Language Support..... 1058
  - 19.4 Intel Device Family Support..... 1058
  - 19.5 Tool Setup..... 1058
    - 19.5.1 Specifying the Intel Quartus Prime Software Version..... 1058
  - 19.6 Synplify Software Generated Files..... 1058
  - 19.7 Design Constraints Support..... 1059
    - 19.7.1 Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script..... 1060
    - 19.7.2 Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software..... 1060
  - 19.8 Simulation and Formal Verification..... 1061
  - 19.9 Synplify Optimization Strategies..... 1061
    - 19.9.1 Using Synplify Premier to Optimize Your Design..... 1062



19.9.2 Using Implementations in Synplify Pro or Premier.....	1062
19.9.3 Timing-Driven Synthesis Settings.....	1062
19.9.4 FSM Compiler.....	1064
19.9.5 Optimization Attributes and Options.....	1065
19.9.6 Intel-Specific Attributes.....	1068
19.10 Guidelines for Intel FPGA IP Cores and Architecture-Specific Features.....	1068
19.10.1 Instantiating Intel FPGA IP Cores with the IP Catalog.....	1069
19.10.2 Including Files for Intel Quartus Prime Placement and Routing Only.....	1073
19.10.3 Inferring Intel FPGA IP Cores from HDL Code.....	1073
19.11 Document Revision History.....	1078



## 1 Introduction to Intel® Quartus® Prime Pro Edition

The Intel® Quartus® Prime software provides a complete design environment for FPGA and SoC designs. The user interface supports easy design entry, fast processing, and straightforward device programming. The Intel Quartus Prime Pro Edition software enables next generation synthesis, physical optimization, design methodologies, and FPGA architectures. The Intel Quartus Prime Pro Edition software provides unique features not available in other Intel Quartus Prime software editions.

The Intel Quartus Prime Pro Edition Compiler is optimized for the latest Intel Arria® 10, Intel Cyclone® 10, and Intel Stratix® 10 devices. The Compiler provides powerful and customizable design processing to achieve the best possible design implementation in silicon. The Intel Quartus Prime software makes it easy for you to focus on your design—not on the design tool. The Intel Quartus Prime Pro Edition software provides unique features not available in other Quartus software products.

**Figure 1. Quartus Prime New Feature Support Matrix**

Software Features	Intel Quartus Prime Standard Edition	Intel Quartus Prime Pro Edition
New Hybrid Placer & Global Router	✓	✓
New Timing Analyzer	✓	✓
New Physical Synthesis	✓	✓
Incremental Fitter Optimization		✓
Interface Planner		✓
New Synthesis Engine		✓
Rapid Recompile		✓
OpenCL		✓
Platform Designer (for Pro Edition)		✓
Partial Reconfiguration		✓
Block-Based Design Flows		✓

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, 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  
9001:2008  
Registered





The modular Compiler streamlines the FPGA development process, and ensures the highest performance for the least effort. The Intel Quartus Prime Pro Edition software provides the following unique features:

- Hyper-Aware Design Flow—use Hyper-Retiming and Fast Forward compilation for the highest performance in Intel Stratix 10 devices.
- Intel Quartus Prime Pro Edition synthesis—integrates new, stricter language parser supporting all major IEEE RTL languages, with enhanced algorithms, and parallel synthesis capabilities. Added support for SystemVerilog 2009.
- Hierarchical project structure—preserves individual post-synthesis, post-placement, and post-place and route results for each design entity. Allows optimization without impacting other partition placement or routing.
- Incremental Fitter Optimizations—run and optimize Fitter stages incrementally. Each Fitter stage generates detailed reports.
- Faster, more accurate I/O placement—plan interface I/O in Interface Planner.
- Platform Designer—builds on the system design and custom IP integration capabilities of Platform Designer. Platform Designer in Intel Quartus Prime Pro Edition introduces hierarchical isolation between system interconnect and IP components.
- Partial Reconfiguration—support reconfiguration of a portion of the Intel Arria 10 FPGA, while the remaining FPGA continues to function.
- Supports block-based design flows, allowing you to preserve and reuse design blocks at various stages of compilation.

*Note:* Intel now refers to the following Intel Quartus Prime tool names:

**Table 1. Intel Quartus Prime Tool Name Updates**

Altera Name	Intel Name
Qsys	Platform Designer
BluePrint	Interface Planner
TimeQuest	Timing Analyzer
EyeQ	Eye Viewer
JNEye	Advanced Link Analyzer

#### Related Links

- [Migrating to Intel Quartus Prime Pro Edition](#) on page 18
- [Upgrade Project Assignments and Constraints](#) on page 19
- [Upgrade IP Cores and Platform Designer Systems](#) on page 24
- [Upgrade Non-Compliant Design RTL](#) on page 25
- [Block-Based Design Flows](#)

## 1.1 Should I Choose the Intel Quartus Prime Pro Edition Software?

Depending on your immediate needs, the Intel Quartus Prime Pro Edition software may be an appropriate choice for your design.



The Intel Quartus Prime Pro Edition software includes many unique features that the Intel Quartus Prime Standard Edition software does not include. However, the Intel Quartus Prime Pro Edition software does not support all features of the Intel Quartus Prime Standard Edition software.

### Selecting a Quartus Prime Edition

Consider the requirements and timeline of your project in determining whether the Intel Quartus Prime Standard Edition or Intel Quartus Prime Pro Edition software is most appropriate for you. Use the following factors to inform your decision:

- The Intel Quartus Prime Pro Edition software supports only Intel Arria 10, Intel Cyclone 10 GX, and Intel Stratix 10 devices. If your design targets any other Intel FPGA device, select the Intel Quartus Prime Standard Edition.
- Select the Intel Quartus Prime Pro Edition if you are beginning a new Intel Arria 10, Intel Cyclone 10 GX, or Intel Stratix 10 design, or if your design requires any unique Intel Quartus Prime Pro Edition features.
- Intel Quartus Prime Pro Edition software does not support the following Intel Quartus Prime Standard Edition features:
  - I/O Timing Analysis
  - NativeLink third party tool integration
  - Video and Image Processing Suite IP Cores
  - Talkback features
  - Various register merging and duplication settings
  - Saving a node-level netlist as .vqm
  - Compare project revisions

### Related Links

- [Managing Projects](#)
- [Design Compilation](#)
- [Creating a Partial Reconfiguration Design](#)

## 1.2 Migrating to Intel Quartus Prime Pro Edition

The Intel Quartus Prime Pro Edition software supports migration of Intel Quartus Prime Standard Edition, Quartus Prime Lite Edition, and Quartus II software projects.

*Note:* The migration steps for Quartus Prime Lite Edition, Intel Quartus Prime Standard Edition, and the Quartus II software are identical. For brevity, this section refers to these design tools collectively as "other Quartus software products."

Migrating to Intel Quartus Prime Pro Edition requires the following changes to other Quartus software product projects:

1. Upgrade project assignments and constraints with equivalent Intel Quartus Prime Pro Edition assignments.
2. Upgrade all Intel FPGA IP core variations and Platform Designer systems in your project.
3. Upgrade design RTL to standards-compliant VHDL, Verilog HDL, or SystemVerilog.



This document describes each migration step in detail.

### 1.2.1 Keep Pro Edition Project Files Separate

The Intel Quartus Prime Pro Edition software does not support project or constraint files from other Quartus software products. Do not place project files from other Quartus software products in the same directory as Intel Quartus Prime Pro Edition project files. In general, use Intel Quartus Prime Pro Edition project files and directories only for Intel Quartus Prime Pro Edition projects, and use other Quartus software product files only with those software tools.

Intel Quartus Prime Pro Edition projects do not support compilation in other Quartus software products, and vice versa. The Intel Quartus Prime Pro Edition software generates an error if the Compiler detects other Quartus software product's features in project files.

Before migrating other Quartus software product projects, click **Project** > **Archive Project** to save a copy of your original project before making modifications for migration.

### 1.2.2 Upgrade Project Assignments and Constraints

Intel Quartus Prime Pro Edition software introduces changes to handling of project assignments and constraints that the Quartus Settings File (.qsf) stores. Upgrade other Quartus software product project assignments and constraints for migration to the Intel Quartus Prime Pro Edition software. Upgrade other Quartus software product assignments with **Assignments** > **Assignment Editor**, by editing the .qsf file directly, or by using a Tcl script.

The following sections detail each type project assignment upgrade that migration requires.

#### Related Links

- [Modify Entity Name Assignments](#) on page 19
- [Resolve Timing Constraint Entity Names](#) on page 20
- [Verify Generated Node Name Assignments](#) on page 20
- [Replace Logic Lock \(Standard\) Regions](#) on page 21
- [Modify Signal Tap Logic Analyzer Files](#) on page 23
- [Remove Unsupported Feature Assignments](#) on page 24

#### 1.2.2.1 Modify Entity Name Assignments

Intel Quartus Prime Pro Edition software supports assignments that include instance names *without* a corresponding entity name.

- "a\_entity:a|b\_entity:b|c\_entity:c" (includes deprecated entity names)
- "a|b|c" (omits deprecated entity names)

While the current version of the Intel Quartus Prime Pro Edition software still *accepts* entity names in the .qsf, the Compiler *ignores* the entity name. The Compiler generates a warning message upon detection of an entity names in the .qsf.

Whenever possible, you should remove entity names from assignments, and discontinue reliance on entity-based assignments. Future versions of the Intel Quartus Prime Pro Edition software may eliminate all support for entity-based assignments.

### 1.2.2.2 Resolve Timing Constraint Entity Names

The Intel Quartus Prime Pro Edition Timing Analyzer honors entity names in Synopsys Design Constraints (.sdc) files.

Use .sdc files from other Quartus software products without modification. However, any scripts that include custom processing of names that the .sdc command returns, such as `get_registers` may require modification. Your scripts must reflect that returned strings do not include entity names.

The .sdc commands respect wildcard patterns containing entity names. Review the Timing Analyzer reports to verify application of all constraints. The following example illustrates differences between functioning and non-functioning .sdc scripts:

```
# Apply a constraint to all registers named "acc" in the entity "counter".
# This constraint functions in both SE and PE, because the SDC
# command always understands wildcard patterns with entity names in them
set_false_path -to [get_registers "counter:*|*acc"]

# This does the same thing, but first it converts all register names to
# strings, which includes entity names by default in the SE
# but excludes them by default in the PE. The regexp will therefore
# fail in PE by default.
#
# This script would also fail in the SE, and earlier
# versions of Quartus II, if entity name display had been disabled
# in the QSF.
set all_reg_strs [query_collection -list -all [get_registers *]]
foreach keeper $all_reg_strs {
    if {[regexp {counter:*|:*acc} $keeper]} {
        set_false_path -to $keeper
    }
}
```

Removal of the entity name processing from .sdc files may not be possible due to complex processing involving node names. Use standard .sdc whenever possible to replace such processing. Alternatively, add the following code to the top and bottom of your script to temporarily re-enable entity name display in the .sdc file:

```
# This script requires that entity names be included
# due to custom name processing
set old_mode [set_project_mode -get_mode_value always_show_entity_name]
set_project_mode -always_show_entity_name on

<... the rest of your script goes here ...>

# Restore the project mode
set_project_mode -always_show_entity_name $old_mode
```

### 1.2.2.3 Verify Generated Node Name Assignments

Intel Quartus Prime synthesis generates and automatically names internal design nodes during processing. The Intel Quartus Prime Pro Edition uses different conventions than other Quartus software products to generate node names during synthesis. When you synthesize your other Quartus software product project in Intel Quartus Prime Pro Edition, the synthesis-generated node names may change. If any



scripts or constraints depend on the synthesis-generated node names, update the scripts or constraints to match the Intel Quartus Prime Pro Edition synthesis node names.

Avoid dependence on synthesis-generated names due to frequent changes in name generation. In addition, verify the names of duplicated registers and PLL clock outputs to ensure compatibility with any script or constraint.

### 1.2.2.4 Replace Logic Lock (Standard) Regions

Intel Quartus Prime Pro Edition software introduces more simplified and flexible Logic Lock constraints, compared with previous Logic Lock regions. You must replace all Logic Lock (Standard) assignments with compatible Logic Lock assignments for migration.

To convert Logic Lock (Standard) regions to Logic Lock regions:

1. Edit the .qsf to delete or comment out all of the following Logic Lock assignments:

```
set_global_assignment -name LL_ENABLED*
set_global_assignment -name LL_AUTO_SIZE*
set_global_assignment -name LL_STATE FLOATING*
set_global_assignment -name LL_RESERVED*
set_global_assignment -name LL_CORE_ONLY*
set_global_assignment -name LL_SECURITY_ROUTING_INTERFACE*
set_global_assignment -name LL_IGNORE_IO_BANK_SECURITY_CONSTRAINT*
set_global_assignment -name LL_PR_REGION*
set_global_assignment -name LL_ROUTING_REGION_EXPANSION_SIZE*
set_global_assignment -name LL_WIDTH*
set_global_assignment -name LL_HEIGHT
set_global_assignment -name LL_ORIGIN
set_instance_assignment -name LL_MEMBER_OF
```

2. Edit the .qsf or click **Tools > Chip Planner** to define new Logic Lock regions. Logic Lock constraint syntax is simplified, for example:

```
set_instance_assignment -name PLACE_REGION "1 1 20 20" -to f101
set_instance_assignment -name RESERVE_PLACE_REGION OFF -to f101
set_instance_assignment -name CORE_ONLY_PLACE_REGION OFF -to f101
```

Compilation fails if synthesis finds other Quartus software product's Logic Lock assignments in a Intel Quartus Prime Pro Edition project. The following table compares other Quartus software product region constraint support with the Intel Quartus Prime Pro Edition software.

**Table 2. Region Constraints Per Edition**

Constraint Type	Logic Lock (Standard) Region Support Other Quartus Software Products	Logic Lock Region Support Intel Quartus Prime Pro Edition
Fixed rectangular, nonrectangular or non-contiguous regions	Full support.	Full support.
Chip Planner entry	Full support.	Full support.
Periphery element assignments	Supported in some instances.	Full support. Use "core-only" regions to exclude the periphery.
Nested ("hierarchical") regions	Supported but separate hierarchy from the user instance tree.	Supported in same hierarchy as user instance tree.

*continued...*



Constraint Type	Logic Lock (Standard) Region Support Other Quartus Software Products	Logic Lock Region Support Intel Quartus Prime Pro Edition
Reserved regions	Limited support for nested or nonrectangular reserved regions. Reserved regions typically cannot cross I/O columns; use non-contiguous regions instead.	Full support for nested and nonrectangular regions. Reserved regions can cross I/O columns without affecting periphery logic if the regions are "core-only".
Routing regions	Limited support via "routing expansion." No support with hierarchical regions.	Full support (including future support for hierarchical regions).
Floating or autosized regions	Full support.	No support.
Region names	Regions have names.	Regions are identified by the instance name of the constrained logic.
Multiple instances in the same region	Full support.	Support for non-reserved regions. Create one region per instance, and then specify the same definition for multiple instances to assign to the same area. Not supported for reserved regions.
Member exclusion	Full support.	No support for arbitrary logic. Use a core-only region to exclude periphery elements. Use non-rectangular regions to include more RAM or DSP columns as needed.

#### 1.2.2.4.1 Logic Lock Region Assignment Examples

These examples show the syntax of Logic Lock region assignments in the .qsf file. Optionally, enter these assignments in the Assignment Editor, the Logic Lock Regions Window, or the Chip Planner.

##### Example 1. Assign Rectangular Logic Lock Region

Assigns a rectangular Logic Lock region to a lower right corner location of (10,10), and an upper right corner of (20,20) inclusive.

```
set_instance_assignment -name PLACE_REGION -to a|b|c "X10 Y10 X20 Y20"
```

##### Example 2. Assign Non-Rectangular Logic Lock Region

Assigns instance with full hierarchical path "x|y|z" to non-rectangular L-shaped Logic Lock region. The software treats each set of four numbers as a new box.

```
set_instance_assignment -name PLACE_REGION -to x|y|z "X10 Y10 X20 Y50; X20 Y10 X50 Y20"
```

##### Example 3. Assign Subordinate Logic Lock Instances

By default, the Intel Quartus Prime software constrains every child instance to the Logic Lock region of its parent. Any constraint to a child instance intersects with the constraint of its ancestors. For example, in the following example, all logic beneath "a|b|c|d" constrains to box (10,10), (15,15), and not (0,0), (15,15). This result occurs because the child constraint intersects with the parent constraint.

```
set_instance_assignment -name PLACE_REGION -to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment -name PLACE_REGION -to a|b|c|d "X0 Y0 X15 Y15"
```



#### Example 4. Assign Multiple Logic Lock Instances

By default, a Logic Lock region constraint allows logic from other instances to share the same region. These assignments place instance `c` and instance `g` in the same location. This strategy is useful if instance `c` and instance `g` are heavily interacting.

```
set_instance_assignment -name PLACE_REGION -to a|b|c "X10 Y10 X20 Y20"  
set_instance_assignment -name PLACE_REGION -to e|f|g "X10 Y10 X20 Y20"
```

#### Example 5. Assigned Reserved Logic Lock Regions

Optionally reserve an entire Logic Lock region for one instance and any of its subordinate instances.

```
set_instance_assignment -name PLACE_REGION -to a|b|c "X10 Y10 X20 Y20"  
set_instance_assignment -name RESERVE_PLACE_REGION -to a|b|c ON  
  
# The following assignment causes an error. The logic in e|f|g is not  
# legally placeable anywhere:  
# set_instance_assignment -name PLACE_REGION -to e|f|g "X10 Y10 X20 Y20"  
  
# The following assignment does *not* cause an error, but is effectively  
# constrained to the box (20,10), (30,20), since the (10,10),(20,20) box is  
# reserved  
# for a|b|c  
set_instance_assignment -name PLACE_REGION -to e|f|g "X10 Y10 X30 Y20"
```

### 1.2.2.5 Modify Signal Tap Logic Analyzer Files

Intel Quartus Prime Pro Edition introduces new methodology for entity names, settings, and assignments. These changes impact the processing of Signal Tap Logic Analyzer Files (`.stp`).

If you migrate a project that includes `.stp` files generated by other Quartus software products, you must make the following changes to migrate to the Intel Quartus Prime Pro Edition:

1. Remove entity names from `.stp` files. The Signal Tap Logic Analyzer allows without error, but ignores, entity names in `.stp` files. Remove entity names from `.stp` files for migration to Intel Quartus Prime Pro Edition:
  - a. Click **View > Node Finder** to locate and remove appropriate nodes. Use Node Finder options to filter on nodes.
  - b. Click **Processing > Start > Start Analysis & Elaboration** to repopulate the database and add valid node names.
2. Remove post-fit nodes. Intel Quartus Prime Pro Edition uses a different post-fit node naming scheme than other Quartus software products.
  - a. Remove post-fit tap node names originating from other Quartus software products.
  - b. Click **View > Node Finder** to locate and remove post-fit nodes. Use Node Finder options to filter on nodes.



- c. Click **Processing** ► **Start Compilation** to repopulate the database and add valid post-fit nodes.
3. Run an initial compilation in Intel Quartus Prime Pro Edition from the GUI. The Compiler automatically removes Signal Tap assignments originating other Quartus software products. Alternatively, from the command-line, run `quartus_stp` once on the project to remove outmoded assignments.  
*Note:* `quartus_stp` introduces no migration impact in the Intel Quartus Prime Pro Edition. Your scripts require no changes to `quartus_stp` for migration.
4. Modify `.sdc` constraints for JTAG. Intel Quartus Prime Pro Edition does not support embedded `.sdc` constraints for JTAG signals. Modify the timing template to suit the design's JTAG driver and board.

### 1.2.2.6 Remove Unsupported Feature Assignments

The Intel Quartus Prime Pro Edition software does not support some feature assignments that other Quartus software products support. Remove the following unsupported feature assignments from other Quartus software product `.qsf` files for migration to the Intel Quartus Prime Pro Edition software.

- Incremental Compilation (partitions)—The current version of the Intel Quartus Prime Pro Edition software does not support Intel Quartus Prime Standard Edition incremental compilation. Remove all incremental compilation feature assignments from other Quartus software product `.qsf` files before migration.
- Intel Quartus Prime Standard Edition Physical synthesis assignments. Intel Quartus Prime Pro Edition software does not support Intel Quartus Prime Standard Edition Physical synthesis assignments. Remove any of the following assignments from the `.qsf` file or design RTL (instance assignments) before migration.

```
PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA  
PHYSICAL_SYNTHESIS_COMBO_LOGIC  
PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION  
PHYSICAL_SYNTHESIS_REGISTER_RETIMING  
PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING  
PHYSICAL_SYNTHESIS_MAP_LOGIC_TO_MEMORY_FOR_AREA
```

### 1.2.3 Upgrade IP Cores and Platform Designer Systems

Upgrade all IP cores and Platform Designer systems in your project for migration to the Intel Quartus Prime Pro Edition software. The Intel Quartus Prime Pro Edition software uses standards-compliant methodology for instantiation and generation of IP cores and Platform Designer systems. Most Intel FPGA IP cores and Platform Designer systems upgrade automatically in the **Upgrade IP Components** dialog box.

Other Quartus software products use a proprietary Verilog configuration scheme within the top level of IP cores and Platform Designer systems for synthesis files. The Intel Quartus Prime Pro Edition does not support this scheme. To upgrade all IP cores and Platform Designer systems in your project, click **Project** ► **Upgrade IP Components**.<sup>(1)</sup>

---

<sup>(1)</sup> For brevity, this section refers to Intel Quartus Prime Standard Edition, Intel Quartus Prime Lite Edition, and the Quartus II software collectively as "other Quartus software products."





**Table 3. IP Core and Platform Designer System Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
<p>IP and Platform Designer system generation use a proprietary Verilog HDL configuration scheme within the top level of IP cores and Platform Designer systems for synthesis files. This proprietary Verilog HDL configuration scheme prevents RTL entities from ambiguous instantiation errors during synthesis. However, these errors may manifest in simulation. Resolving this issue requires writing a Verilog HDL configuration to disambiguate the instantiation, delete the duplicate entity from the project, or rename one of the conflicting entities. Intel Quartus Prime Pro Edition IP strategy resolves these issues.</p>	<p>IP and Platform Designer system generation does not use proprietary Verilog HDL configurations. The compilation library scheme changes in the following ways:</p> <ul style="list-style-type: none"> <li>• Compiles all variants of an IP core into the same compilation library across the entire project. Intel Quartus Prime Pro Edition identically names IP cores with identical functionality and parameterization to avoid ambiguous entity instantiation errors. For example, the files for everyIntel Arria 10 PCI Express* IP core variant compile into the altera_pcie_a10_hip_151 compilation library.</li> <li>• Simulation and synthesis file sets for IP cores and systems instantiate entities in the same manner.</li> <li>• The generated RTL directory structure now matches the compilation library structure.</li> </ul>

**Note:** For complete information on upgrading IP cores, refer to *Managing Intel Quartus Prime Projects*.

**Related Links**

- [Introduction to Intel FPGA IP Cores](#)
- [Upgrading IP Cores](#)
- [Managing Intel Quartus Prime Projects](#) on page 32

**1.2.4 Upgrade Non-Compliant Design RTL**

The Intel Quartus Prime Pro Edition software introduces a new synthesis engine (`quartus_syn` executable).

The `quartus_syn` synthesis enforces stricter industry-standard HDL structures and supports the following enhancements in this release:

- More robust support for SystemVerilog
- Improved support for VHDL2008
- New RAM inference engine infers RAMs from GENERATE statements or array of integers
- Stricter syntax/semantics check for improved compatibility with other EDA tools

Account for these synthesis differences in existing RTL code by ensuring that your design uses standards-compliant VHDL, Verilog HDL, or SystemVerilog. The Compiler generates errors when processing non-compliant RTL. Use the guidelines in this section to modify existing RTL for compatibility with the Intel Quartus Prime Pro Edition synthesis.

**Related Links**

- [Verify Verilog Compilation Unit](#) on page 26
- [Update Entity Auto-Discovery](#) on page 27
- [Ensure Distinct VHDL Namespace for Each Library](#) on page 27
- [Remove Unsupported Parameter Passing](#) on page 27
- [Remove Unsized Constant from WYSIWYG Instantiation](#) on page 28



- [Remove Non-Standard Pragmas](#) on page 28
- [Declare Objects Before Initial Values](#) on page 28
- [Confine SystemVerilog Features to SystemVerilog Files](#) on page 29
- [Avoid Assignment Mixing in Always Blocks](#) on page 29
- [Avoid Unconnected, Non-Existent Ports](#) on page 30
- [Avoid Illegal Parameter Ranges](#) on page 30
- [Update Verilog HDL and VHDL Type Mapping](#) on page 30

### 1.2.4.1 Verify Verilog Compilation Unit

Intel Quartus Prime Pro Edition synthesis uses a different method to define the compilation unit. The Verilog LRM defines the concept of compilation unit as “a collection of one or more Verilog source files compiled together” forming the compilation-unit scope. Items visible only in the compilation-unit scope include macros, global declarations, and default net types. The contents of included files become part of the compilation unit of the parent file. Modules, primitives, programs, interfaces, and packages are visible in all compilation units. Ensure that your RTL accommodates these changes.

**Table 4. Verilog Compilation Unit Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
Synthesis in other Quartus software products follows the Multi-file compilation unit (MFCU) method to select compilation unit files. In MFCU, all files compile in the same compilation unit. Global definitions and directives are visible in all files. However, the default net type is reset at the start of each file.	Intel Quartus Prime Pro Edition synthesis follows the Single-file compilation unit (SFCU) method to select compilation unit files. In SFCU, each file is a compilation unit, file order is irrelevant, and the macro is only defined until the end of the file.

**Note:** You can optionally change the MFCU mode using the following assignment:  
`set_global_assignment -name VERILOG_CU_MODE MFCU`

#### 1.2.4.1.1 Verilog HDL Configuration Instantiation

Intel Quartus Prime Pro Edition synthesis requires instantiation of the Verilog HDL configuration, and not the module. In other Quartus software products, synthesis automatically finds any Verilog HDL configuration relating to a module that you instantiate. The Verilog HDL configuration then instantiates the design.

If your top-level entity is a Verilog HDL configuration, set the Verilog HDL configuration, rather than the module, as the top-level entity.

**Table 5. Verilog HDL Configuration Instantiation**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
From the Example RTL, synthesis automatically finds the <code>mid_config</code> Verilog HDL configuration relating to the instantiated module.	From the Example RTL, synthesis does not find the <code>mid_config</code> Verilog HDL configuration. You must instantiate the Verilog HDL configuration directly.
Example RTL: <pre>config mid_config; design good_lib.mid; instance mid.sub_inst use good_lib.sub; endconfig</pre>	
<i>continued...</i>	



Other Quartus Software Products	Intel Quartus Prime Pro Edition
<pre> module test (input a1, output b); mid_config mid_inst ( .a1(a1), .b(b)); // in other Quartus products preceding line would have been: //mid mid_inst ( .a1(a1), .b(b)); endmodule  module mid (input a1, output b); sub_sub_inst (.a1(a1), .b(b)); endmodule </pre>	

### 1.2.4.2 Update Entity Auto-Discovery

All editions of the Intel Quartus Prime and Quartus II software search your project directory for undefined entities. For example, if you instantiate entity “sub” in your design without specifying “sub” as a design file in the Quartus Settings File (.qsf), synthesis searches for sub.v, sub.vhd, and so on. However, Intel Quartus Prime Pro Edition performs auto-discovery at a different stage in the flow. Ensure that your RTL code accommodates these auto-discovery changes.

**Table 6. Entity Auto-Discovery Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
Always automatically searches your project directory and search path for undefined entities.	Always automatically searches your project directory and search path for undefined entities. Intel Quartus Prime Pro Edition synthesis performs auto-discovery earlier in the flow than other Quartus software products. This results in discovery of more syntax errors. Optionally disable auto-discovery with the following .qsf assignment: set_global_assignment -name AUTO_DISCOVER_AND_SORT OFF

### 1.2.4.3 Ensure Distinct VHDL Namespace for Each Library

Intel Quartus Prime Pro Edition synthesis requires that VHDL namespaces are distinct for each library. The stricter library binding requirement complies with VHDL language specifications and results in deterministic behavior. This benefits team-based projects by avoiding unintentional name collisions. Confirm that your RTL respects this change.

**Table 7. VHDL Namespace Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
For the Example RTL, the analyzer searches all libraries in an unspecified order until the analyzer finds package utilities_pack and uses items from that package. If another library, for example projectLib also contains utilities_pack, the analyzer may use this library instead of myLib.utilities_pack if found before the analyzer searches myLib.	For the Example RTL, the analyzer uses the specific utilities_pack in myLib. If utilities_pack does not exist in library myLib, the analyzer generates an error.
Example RTL: <pre> library myLib; use myLib.utilities_pack.all; </pre>	

### 1.2.4.4 Remove Unsupported Parameter Passing

Intel Quartus Prime Pro Edition synthesis does not support parameter passing using set\_parameter in the .qsf. Synthesis in other Quartus software products supports passing parameters with this method. Except for the top-level of the design where permitted, ensure that your RTL does not depend on this type of parameter passing.

**Table 8. SystemVerilog Feature Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
<p>From the Example RTL, synthesis overwrites the value of parameter SIZE in the instance of <code>my_ram</code> instantiated from entity <code>mid-level</code>.</p>	<p>From the Example RTL, synthesis generates a syntax error for detection of parameter passing assignments in the <code>.qsf</code>. Specify parameters in the RTL. The following example shows the supported top-level parameter passing format. This example applies only to the top-level and sets a value of 4 to parameter N:</p> <pre data-bbox="683 478 1386 510">set_parameter -name N 4</pre>
<p>Example RTL:</p> <pre data-bbox="256 573 1386 604">set_parameter -entity mid_level -to my_ram -name SIZE 16</pre>	

### 1.2.4.5 Remove Unsized Constant from WYSIWYG Instantiation

Intel Quartus Prime Pro Edition synthesis does not allow use of an unsized constant for WYSIWYG instantiation. Synthesis in other Quartus software products allows use of SystemVerilog (`.sv`) unsized constants when instantiating a WYSIWYG in a `.v` file.

Intel Quartus Prime Pro Edition synthesis allows use of unsized constants in `.sv` files for uses other than WYSIWYG instantiation. Ensure that your RTL code does not use unsized constants for WYSIWYG instantiation. For example, specify a sized literal, such as `2'b11`, rather than `'1`.

### 1.2.4.6 Remove Non-Standard Pragmas

Intel Quartus Prime Pro Edition synthesis does not support the `vhdl(verilog)_input_version` pragma or the `library` pragma. Synthesis in other Quartus software products supports these pragmas. Remove any use of the pragmas from RTL for Intel Quartus Prime Pro Edition migration. Use the following guidelines to implement the pragma functionality in Intel Quartus Prime Pro Edition:

- `vhdl(verilog)_input_version` Pragma—allows change to the input version in the middle of an input file. For example, to change VHDL 1993 to VHDL 2008. For Intel Quartus Prime Pro Edition migration, specify the input version for each file in the `.qsf`.
- `library` Pragma—allows changes to the VHDL library into which files compile. For Intel Quartus Prime Pro Edition migration, specify the compilation library in the `.qsf`.

### 1.2.4.7 Declare Objects Before Initial Values

Intel Quartus Prime Pro Edition synthesis requires declaration of objects before initial value. Ensure that your RTL declares objects before initial value. Other Quartus software products allow declaration of initial value prior to declaration of the object.



**Table 9. Object Declaration Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
From the Example RTL, synthesis initializes the output <code>p_prog_io1</code> with the value of <code>p_progiol_reg</code> , even though the register declaration occurs in Line 2.	From the Example RTL, synthesis generates a syntax error when you specify initial values before declaring the register.
Example RTL:	
<pre>1 output p_prog_io1 = p_prog_io1_reg; 2 reg p_prog_io1_reg;</pre>	

### 1.2.4.8 Confine SystemVerilog Features to SystemVerilog Files

Intel Quartus Prime Pro Edition synthesis does not allow SystemVerilog features in Verilog HDL files. Other Quartus software products allow use of a subset of SystemVerilog (.sv) features in Verilog HDL (.v) design files. To avoid syntax errors in Intel Quartus Prime Pro Edition, allow only SystemVerilog features in Verilog HDL files.

To use SystemVerilog features in your existing Verilog HDL files, rename your Verilog HDL (.v) files as SystemVerilog (.sv) files. Alternatively, you can set the file type in the .qsf, as shown in the following example:

```
set_global_assignment -name SYSTEMVERILOG_FILE <file>.v
```

**Table 10. SystemVerilog Feature Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
From the Example RTL, synthesis interprets <code>\$clog2</code> in a .v file, even though the Verilog LRM does not define the <code>\$clog2</code> feature. Other Quartus software products allow other SystemVerilog features in .v files.	From the Example RTL, synthesis generates a syntax error for detection of any non-Verilog HDL construct in .v files. Intel Quartus Prime Pro Edition synthesis honors SystemVerilog features only in .sv files.
Example RTL:	
<pre>localparam num_mem_locations = 1050; wire mem_addr [ \$clog2(num_mem_locations)-1 : 0];</pre>	

### 1.2.4.9 Avoid Assignment Mixing in Always Blocks

Intel Quartus Prime Pro Edition synthesis does not allow mixed use of blocking and non-blocking assignments within ALWAYS blocks. Other Quartus software products allow mixed use of blocking and non-blocking assignments within ALWAYS blocks. To avoid syntax errors, ensure that ALWAYS block assignments are of the same type for Intel Quartus Prime Pro Edition migration.

**Table 11. ALWAYS Block Assignment Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
Synthesis honors the mixed blocking and non-blocking assignments, although the Verilog Language Specification no longer supports this construct.	Synthesis generates a syntax error for detection of mixed blocking and non-blocking assignments within an ALWAYS block.



### 1.2.4.10 Avoid Unconnected, Non-Existent Ports

Intel Quartus Prime Pro Edition synthesis requires that a port exists in the module prior to instantiation and naming. Other Quartus software products allow you to instantiate and name an unconnected port that does not exist in the module. Modify your RTL to match this requirement.

To avoid syntax errors, remove all unconnected and non-existent ports for Intel Quartus Prime Pro Edition migration.

**Table 12. Unconnected, Non-Existent Port Differences**

Other Quartus Software Products	Intel Quartus Prime Pro Edition
Synthesis allows you to instantiate and name unconnected or non-existent ports that do not exist on the module.	Synthesis generates a syntax error for detection of mixed blocking and non-blocking assignments within an ALWAYS block.

### 1.2.4.11 Avoid Illegal Parameter Ranges

Intel Quartus Prime Pro Edition synthesis generates an error for detection of constant numeric (integer or floating point) parameter values that exceed the language specification. Other Quartus software products allow constant numeric (integer or floating point) values for parameters that exceed the language specifications. To avoid syntax errors, ensure that constant numeric (integer or floating point) values for parameters conform to the language specifications.

### 1.2.4.12 Update Verilog HDL and VHDL Type Mapping

Intel Quartus Prime Pro Edition synthesis requires that you use 0 for "false" and 1 for "true" in Verilog HDL files (.v). Other Quartus software products map "true" and "false" strings in Verilog HDL to TRUE and FALSE Boolean values in VHDL. Intel Quartus Prime Pro Edition synthesis generates an error for detection of non-Verilog HDL constructs in .v files. To avoid syntax errors, ensure that your RTL accommodates these standards.

## 1.3 Document Revision History

**Table 13. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"><li>Described Intel Quartus Prime tool name updates for Platform Designer (Qsys), Interface Planner (Blueprint), Timing Analyzer (TimeQuest), Eye Viewer (EyeQ), and Advanced Link Analyzer (Advanced Link Analyzer).</li><li>Added Verilog HDL Macro example.</li><li>Updated for latest Intel branding conventions.</li></ul>
2017.05.08	17.0.0	<ul style="list-style-type: none"><li>Removed statement about limitations for safe state machines. The Compiler supports safe state machines. State machine inference is enabled by default.</li><li>Added reference to Block-Based Design Flows.</li><li>Removed procedure on manual dynamic synthesis report generation. The Compiler automatically generates dynamic synthesis reports when enabled.</li></ul>

*continued...*



Date	Version	Changes
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>• Implemented Intel rebranding.</li> <li>• Added reference to Partial Reconfiguration support.</li> <li>• Added to list of Intel Quartus Prime Standard Edition features unsupported by Intel Quartus Prime Pro Edition.</li> <li>• Added topic on Safe State Machine encoding.</li> <li>• Described unsupported Intel Quartus Prime Standard Edition physical synthesis options.</li> <li>• Removed deprecated <b>Per-Stage Compilation (Beta)</b> Compilation Flow.</li> <li>• Changed title from "Remove Filling Vectors" to "Remove Unsized Constant".</li> </ul>
2016.05.03	16.0.0	<ul style="list-style-type: none"> <li>• Removed software beta status and revised feature set.</li> <li>• Added topic on Safe State Machine encoding.</li> <li>• Added Generating Dynamic Synthesis Reports.</li> <li>• Corrected statement about Verilog Compilation Unit.</li> <li>• Corrected typo in Modify Entity Name Assignments.</li> <li>• Added description of Fitter Plan, Place and Route stages, reporting, and optimization.</li> <li>• Added <b>Per-Stage Compilation (Beta)</b> Compilation Flow.</li> <li>• Added Platform Designer information.</li> <li>• Added OpenCL and Signal Tap with routing preservation as unique Pro Edition features.</li> <li>• Clarified limitations for multiple Logic Lock instances in the same region.</li> </ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>• First version of document.</li> </ul>

### Related Links

#### [Documentation Archive](#)

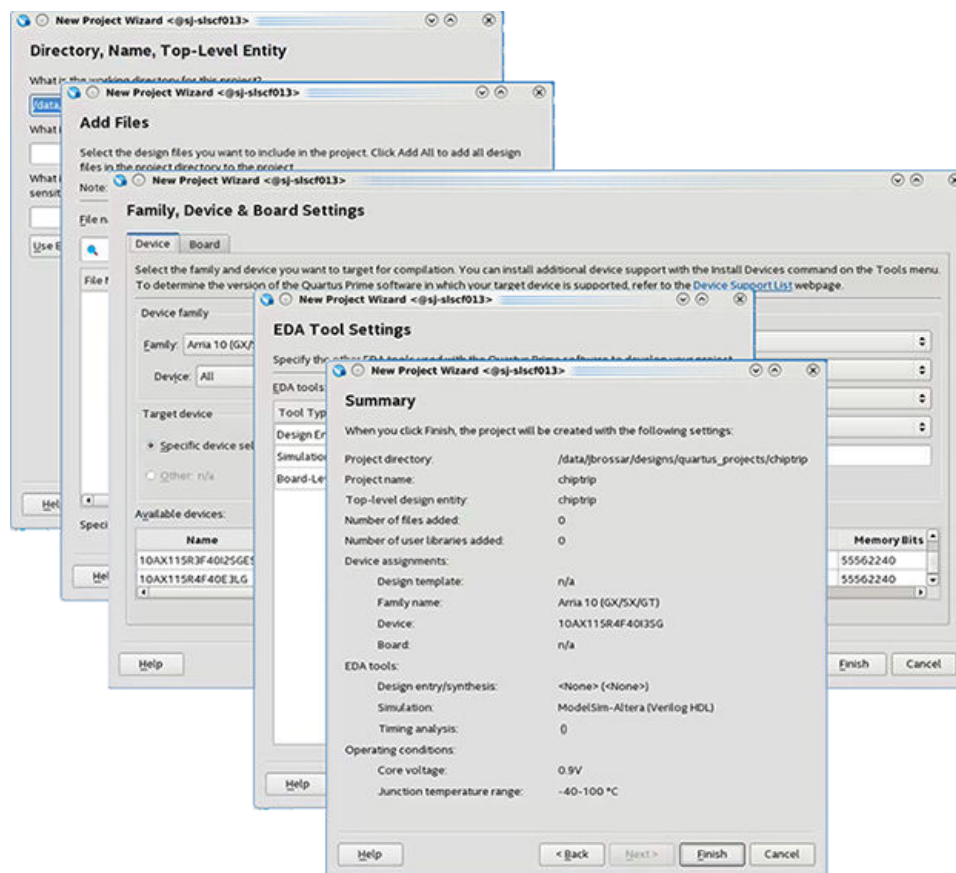
For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

## 2 Managing Intel Quartus Prime Projects

The Intel Quartus Prime software organizes and manages the elements of your design within a *project*.

Click **File > New Project Wizard** to quickly setup and create a new design project.

**Figure 2. New Project Wizard**



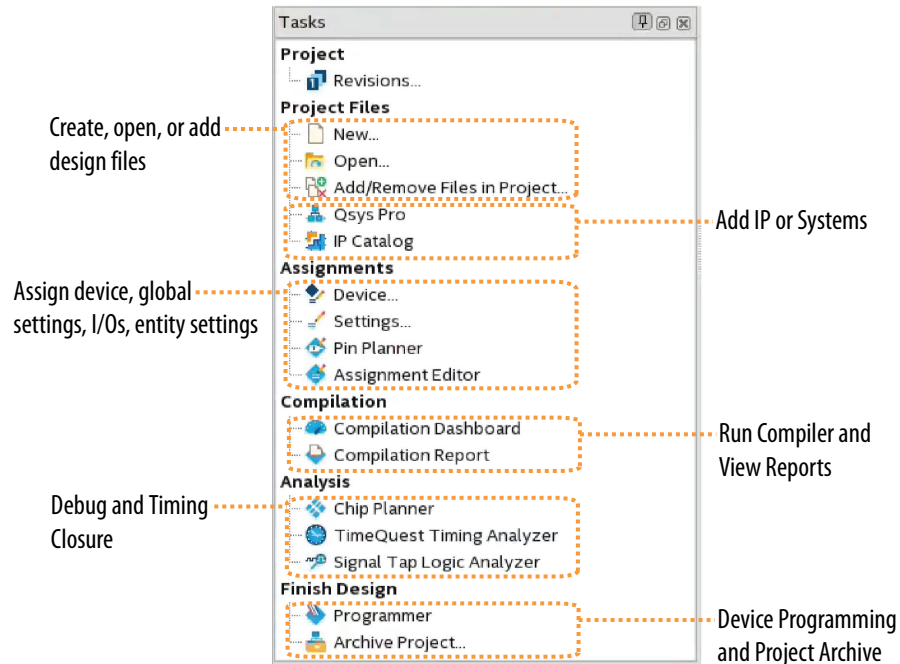
When you open a project, a unified GUI displays integrated project information. The project encapsulates information about your design hierarchy, libraries, constraints, and project settings.





**Figure 3. Project Tasks Pane**

Use the **Tasks** pane for immediate access to all Intel Quartus Prime project settings.



You can save multiple revisions of your project to experiment with settings that achieve your design goals. Intel Quartus Prime projects support team-based, distributed work flows and a scripting interface.

## 2.1 Understanding Intel Quartus Prime Projects

The Intel Quartus Prime software organizes your FPGA design work within a project. A single Intel Quartus Prime Project File (.qpf) represents each design project. The text-based .qpf references the Intel Quartus Prime Settings File (.qsf). The .qsf references the project's design, constraint, and IP files, and stores and project-wide or entity-specific settings that you specify in the GUI. The Intel Quartus Prime organizes and maintains these various project files.

**Table 14. Intel Quartus Prime Project Files**

File Type	Contains	To Edit	Format
Project file	Project and revision name	<b>File &gt; New Project Wizard</b>	Intel Quartus Prime Project File (.qpf)
Project settings	Lists design files, entity settings, target device, synthesis directives, placement constraints	<b>Assignments &gt; Settings</b>	Intel Quartus Prime Settings File (.qsf)

*continued...*



File Type	Contains	To Edit	Format
Timing constraints	Clock properties, exceptions, setup/hold	<b>Tools &gt; Timing Analyzer</b>	Synopsys Design Constraints File (.sdc)
Logic design files	RTL and other design source files	<b>File &gt; New</b>	All supported HDL files
Programming files	Device programming image and information	<b>Tools &gt; Programmer</b>	SRAM Object File (.sof) Programmer Object File (.pof)
Project library	Project and global library information	<b>Tools &gt; Options &gt; Libraries</b>	.qsf (project) quartus2.ini (global)
IP core files	IP core variation parameterization	<b>Tools &gt; IP Catalog</b>	Intel Quartus Prime IP File (.ip)
Platform Designer system files	Platform Designer system and IP core files	<b>Tools &gt; Platform Designer</b>	Platform Designer System File (.qsys)
EDA tool files	Generated for third-party EDA tools	<b>Assignments &gt; Settings &gt; EDA Tool Settings</b>	Verilog Output File (.vo) VHDL Output File (.vho) Verilog Quartus Mapping File (.vqm)
Archive files	Complete project as single compressed file	<b>Project &gt; Archive Project</b>	Intel Quartus Prime Archive File (.qar)

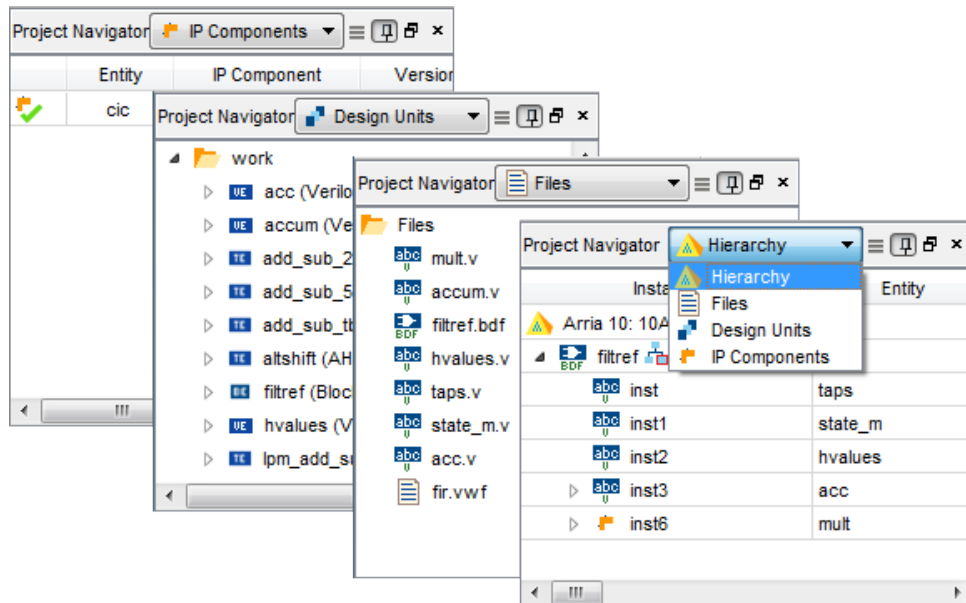
## 2.2 Viewing Basic Project Information

View basic information about your project in the Project Navigator, Compilation Dashboard, Report panel, and **Messages** window. View project elements in the **Project Navigator (View > Project Navigator)**. The **Project Navigator** displays key project information, such as design files, IP components, and your project hierarchy. Use the **Project Navigator** to locate and perform actions of the elements of your project. To access the tabs of the Project Navigator, click the toggle control at the top of the **Project Navigator** window.

**Table 15. Project Navigator Tabs**

Project Navigator Tab	Description
<b>Files</b>	Lists all design files in the current project. Right-click design files in this tab to run these commands: <ul style="list-style-type: none"> <li>• <b>Open</b> the file</li> <li>• <b>Remove</b> the file from project</li> <li>• View file <b>Properties</b></li> </ul>
<b>Hierarchy</b>	Provides a visual representation of the project hierarchy, specific resource usage information, and device and device family information. Right-click items in the hierarchy to <b>Locate</b> , <b>Set as Top-Level Entity</b> , or define Logic Lock regions or design partitions.
<b>Design Units</b>	Displays the design units in the project. Right-click a design unit to <b>Locate in Design File</b> .
<b>IP Components</b>	Displays the design files that make up the IP instantiated in the project, including Intel FPGA IP, Platform Designer components, and third-party IP. Click <b>Launch IP Upgrade Tool</b> from this tab to upgrade outdated IP components. Right-click any IP component to <b>Edit in Parameter Editor</b> .

Figure 4. Project Navigator Hierarchy, Files, Design Units, and IP Components Tabs



## 2.2.1 Viewing Project Reports

The Compilation Report panel updates dynamically to display detailed reports during project processing.

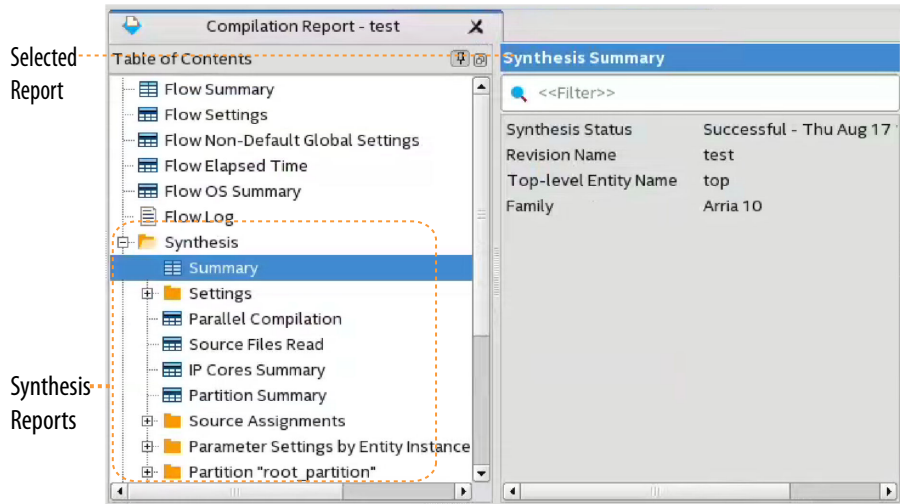
To access the Compilation Report, click (**Processing** ► **Compilation Report**).

*Note:* You can also access the Compilation Report from the Compilation Dashboard (**Processing** ► **Compilation Dashboard**).

- Synthesis reports
- Fitter reports
- Timing analysis reports
- Power analysis reports
- Signal integrity reports

Analyze the detailed project information in these reports to determine correct implementation. Right-click report data to locate and edit the source in project files.

Figure 5. Compilation Report



**Related Links**

[List of Compilation Reports](#)

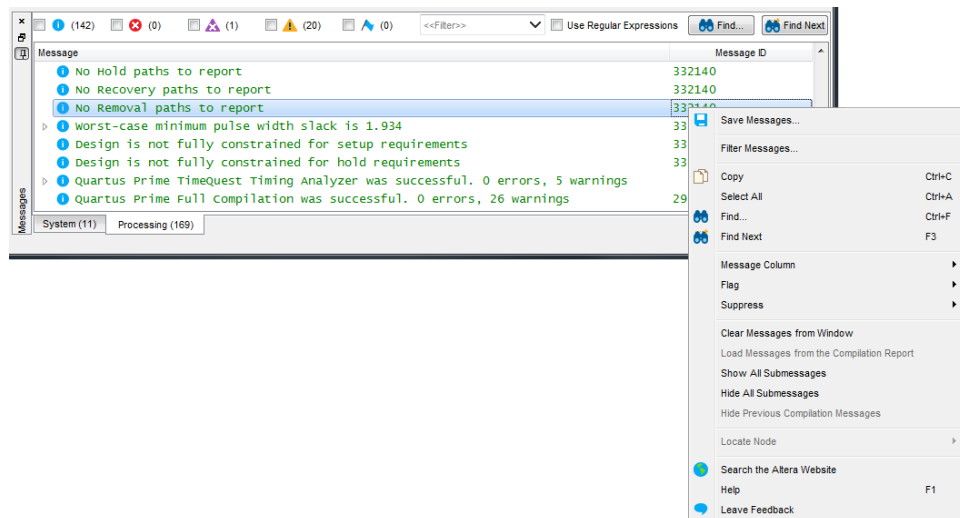
**2.2.2 Viewing Project Messages**

The Messages window (**View ► Messages**) displays information, warning, and error messages about Intel Quartus Prime processes. Right-click messages to locate the source or get message help.

- **Processing** tab—displays messages from the most recent process
- **System** tab—displays messages unrelated to design processing
- **Search**—locates specific messages

Messages are written to stdout when you use command-line executables.

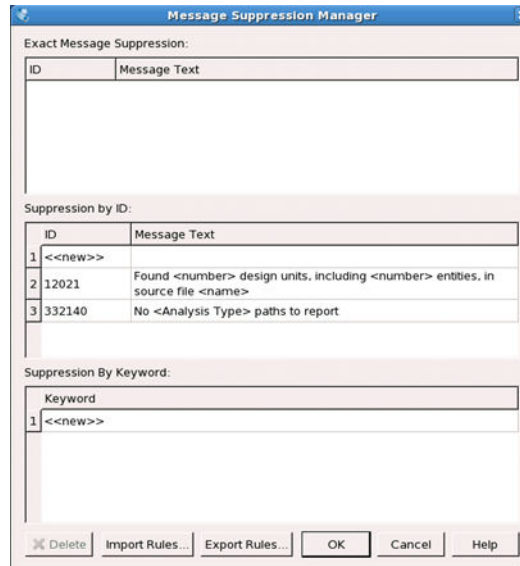
Figure 6. Messages Window





You can suppress display of unimportant messages so they do not obscure valid messages.

**Figure 7. Message Suppression by Message ID Number**



### 2.2.2.1 Suppressing Messages

Suppress any messages that you do not want to view. To suppress messages, right-click a message and choose any of the following:

- **Suppress Message**—suppresses all messages matching exact text
- **Suppress Messages with Matching ID**—suppresses all messages matching the message ID number, ignoring variables
- **Suppress Messages with Matching Keyword**—suppresses all messages matching keyword or hierarchy

### 2.2.2.2 Message Suppression Guidelines

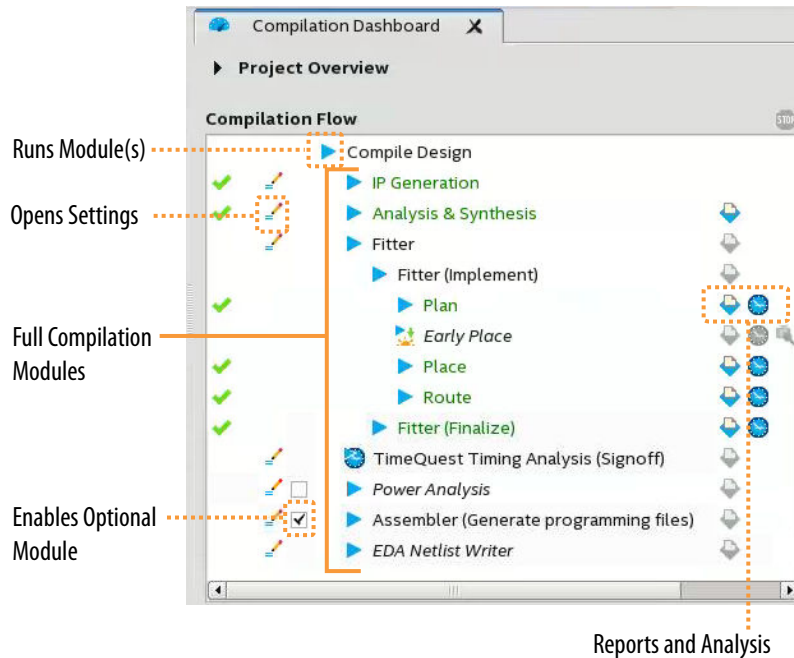
- You cannot suppress error or Intel legal agreement messages.
- Suppressing a message also suppresses any submessages.
- Message suppression is project revision-specific. Derivative project revisions inherit any suppression.
- You cannot edit messages or suppression rules during compilation.

## 2.3 Using the Compilation Dashboard

The Compilation Dashboard provides an overview of your project, and lets you change project settings, compile your design, and view reports for each compilation stage.

The Compilation Dashboard appears by default when you open a project. To open the Compilation Dashboard manually, click **Compilation Dashboard** in the Tasks window. You can also access the Compilation Report from the Compilation Dashboard.

Figure 8. Compilation Dashboard



## 2.4 Project Management Best Practices

The Intel Quartus Prime software provides various options for setting up a project. The following best practices help ensure efficient management and portability of your project files.

### Setting and Project File Best Practices

- Be very careful if you edit any Intel Quartus Prime data files, such as the Intel Quartus Prime Project File (.qpf), Intel Quartus Prime Settings File (.qsf), Quartus IP File (.qip), or Platform Designer System File (.qsys). Typos in these files can cause software errors. For example, the software may ignore settings and assignments.

Every Intel Quartus Prime project revision automatically includes a supporting .qpf that preserves various project settings and constraints that you enter in the GUI or add with Tcl commands. This file contains basic information about the current software version, date, and project-wide and entity level settings. Due to dependencies between the .qpf and .qsf, avoid manually editing .qsf files.

- Do not compile multiple projects into the same directory. Instead, use a separate directory for each project.
- By default, the Intel Quartus Prime software saves all project output files, such as Text-Format Report Files (.rpt), in the project directory. Instead of manually moving project output files, change your project compilation settings to save them in a separate directory.

To save these files into a different directory choose **Assignments > Settings > Compilation Process Settings**. Turn on **Save project output files in specified directory** and specify a directory for the output files.



## Project Archive and Source Control Best Practices

Click **Project > Archive Project** to archive your project for revision control.

As you develop your design, your Intel Quartus Prime project directory contains a variety of source and settings files, compilation database files, output, and report files. You can archive these files using the Archive feature and save the archive for later use or place it under revision control.

1. Choose **Project > Archive Project > Advanced** to open the **Advanced Archive Settings** dialog box.
2. Choose a file set to archive.
3. Add additional files by clicking **Add** (optional).

To restore your archived project, choose **Project > Restore Archived Project**. Restore your project into a new, empty directory.

## IP Core Best Practices

- Do not manually edit or write your own `.qsys`, `.ip`, or `.qip` file. Use the Intel Quartus Prime software tools to create and edit these files.  
*Note:* When generating IP cores, do not generate files into a directory that has a space in the directory name or path. Spaces are not legal characters for IP core paths or names.
- When you generate an IP core using the IP Catalog, the Intel Quartus Prime software generates a `.qsys` (for Platform Designer-generated IP cores) or a `.ip` file (for Intel Quartus Prime Pro Edition) or a `.qip` file. The Intel Quartus Prime Pro Edition software automatically adds the generated `.ip` to your project. In the Intel Quartus Prime Standard Edition software, add the `.qip` to your project. Do not add the parameter editor generated file (`.v` or `.vhd`) to your design without the `.qsys` or `.qip` file. Otherwise, you cannot use the IP upgrade or IP parameter editor feature.
- Plan your directory structure ahead of time. Do not change the relative path between a `.qsys` file and its generation output directory. If you must move the `.qsys` file, ensure that the generation output directory remains with the `.qsys` file.
- Do not add IP core files directly from the **/quartus/libraries/megafunctions** directory in your project. Otherwise, you must update the files for each subsequent software release. Instead, use the IP Catalog and then add the `.qip` to your project.



- Do not use IP files that the Intel Quartus Prime software generates for RAM or FIFO blocks targeting older device families (even though the Intel Quartus Prime software does not issue an error). The RAM blocks that Intel Quartus Prime generates for older device families are not optimized for the latest device families.
- When generating a ROM function, save the resulting `.mif` or `.hex` file in the same folder as the corresponding IP core's `.qsys` or `.qip` file. For example, moving all of your project's `.mif` or `.hex` files to the same directory causes relative path problems after archiving the design.
- Always use the Intel Quartus Prime `ip-setup-simulation` and `ip-make-simscript` utilities to generate simulation scripts for each IP core or Platform Designer system in your design. These utilities produce a single simulation script that does not require manual update for upgrades to Intel Quartus Prime software or IP versions.

### Related Links

[Generating a Combined Simulator Setup Script](#) on page 386

## 2.5 Managing Project Settings

The New Project Wizard guides you to make initial project settings when you setup a new project. Optimizing project settings helps the Compiler to generate programming files that meet or exceed your specifications.

On the **Tasks** pane, click **Settings** to access global project settings, including:

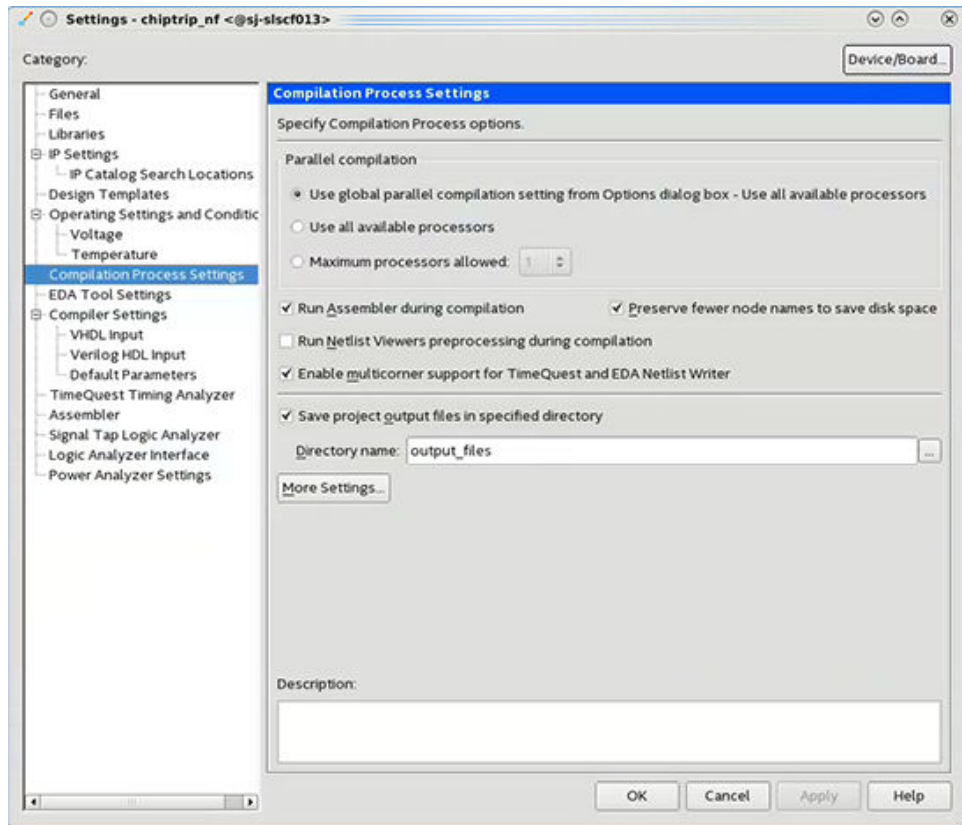
- Project files list
- Synthesis directives and constraints
- Logic options and compiler effort levels
- Placement constraints
- Timing constraint files
- Operating temperature limits and conditions
- File generation for other EDA tools
- Target a device (click **Device**)
- Target a development kit

The `.qsf` stores each project revision's project settings. The Intel Quartus Prime Default Settings File (`<revision name>_assignment_defaults.qdf`) stores the default settings and constraints for each new project revision.



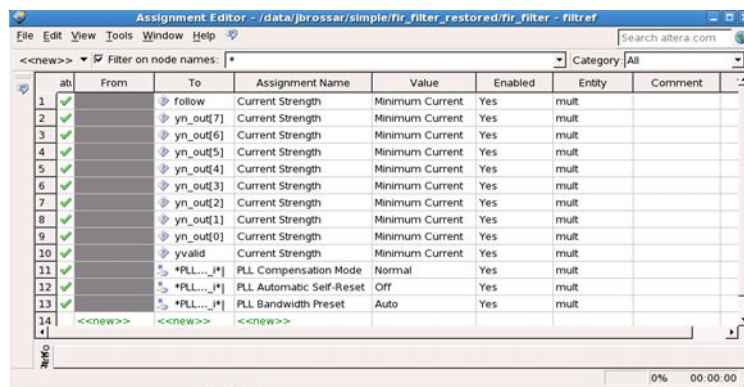


Figure 9. Settings Dialog Box for Global Project Settings



The Assignment Editor (**Tools > Assignment Editor**) provides a spreadsheet-like interface for assigning all instance-specific settings and constraints.

Figure 10. Assignment Editor Spreadsheet



## 2.5.1 Optimizing Project Settings

Optimize project settings to meet your design goals. The Intel Quartus Prime Design Space Explorer II iteratively compiles your project with various setting combinations to find the optimal setting for your goals. Alternatively, you can create a project revision or project copy to manually compare various project settings and design combinations.

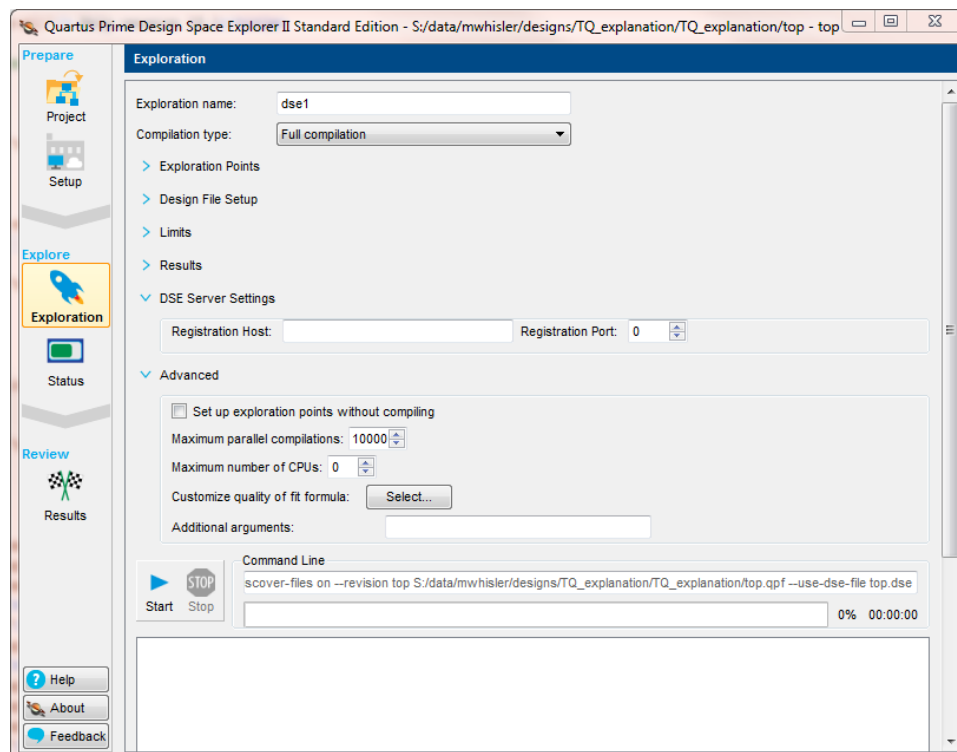
The Intel Quartus Prime software includes several advisors to help you optimize your design and reduce compilation time. The advisors listed in the **Tools > Advisors** menu can provide recommendations based on your project settings and design constraints.

### 2.5.1.1 Optimizing with Design Space Explorer II

Use Design Space Explorer II (**Tools > Launch Design Space Explorer II**) to find optimal project settings for resource, performance, or power optimization goals. Design Space Explorer II (DSE II) processes your design using various setting and constraint combinations, and reports the best settings for your design.

DSE II attempts multiple seeds to identify one meeting your requirements. DSE II can run different compilations on multiple computers in parallel to streamline timing closure.

**Figure 11. Design Space Explorer II**





### 2.5.1.2 Optimizing with Project Revisions

You can save multiple, named project revisions within your Intel Quartus Prime project (**Project > Revisions**).

Each revision captures a unique set of project settings and constraints, but does not capture any logic design file changes. Use revisions to experiment with different settings while preserving the original. Optimize different revisions for various applications. Use revisions for the following:

- Create a unique revision to optimize a design for different criteria, such as by area in one revision and by  $f_{MAX}$  in another revision.
- When you create a new revision the default Intel Quartus Prime settings initially apply.
- Create a revision of a revision to experiment with settings and constraints. The child revision includes all the assignments and settings of the parent revision.

You create, delete, and edit revisions in the **Revisions** dialog box. Each time you create a new project revision, the Intel Quartus Prime software creates a new `.qsf` using the revision name.

### 2.5.1.3 Copying Your Project

Click **Project > Copy Project** to create a separate copy of your project, rather than just a revision within the same project.

The project copy includes all design files, any `.qsf` files, and project revisions. Use this technique to optimize project copies for different applications. For example, optimize one project to interface with a 32-bit data bus, and optimize a project copy to interface with a 64-bit data bus.

### 2.5.1.4 Copy (Back-Annotate) Compiler Assignments

The Compiler maps the elements of your design to specific device and resource during fitting. Following compilation, you can copy the Compiler's device and resource assignments to the `.qsf` to preserve that same implementation in subsequent compilations.

Click **Assignments > Back-Annotate Assignments** to apply the device resource assignments to the `.qsf`. Select the back-annotation type in the **Back-annotation type** list.

## 2.6 Managing Logic Design Files

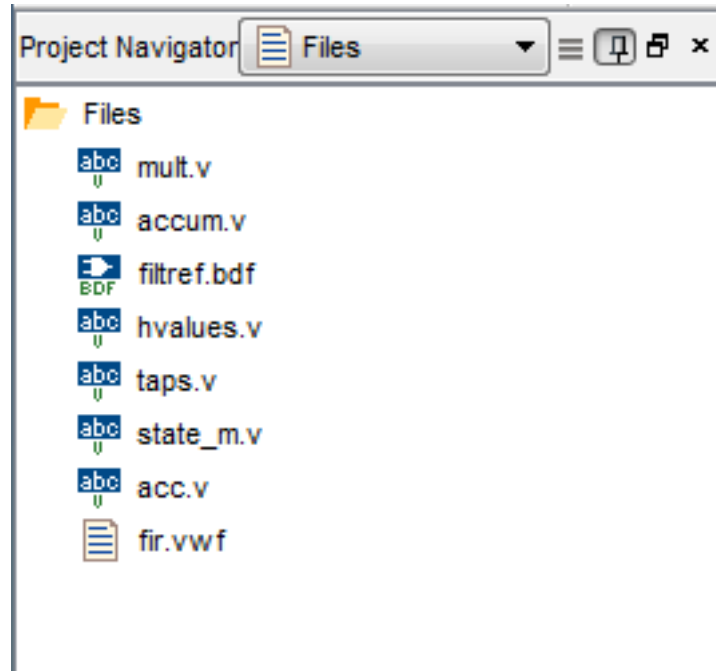
The Intel Quartus Prime software helps you create and manage the logic design files in your project. Logic design files contain the logic that implements your design. When you add a logic design file to the project, the Compiler automatically compiles that file as part of the project. The Compiler synthesizes your logic design files to generate programming files for your target device.

The Intel Quartus Prime software includes full-featured schematic and text editors, as well as HDL templates to accelerate your design work. The Intel Quartus Prime software supports VHDL Design Files (`.vhd`), Verilog HDL Design Files (`.v`),

SystemVerilog (.sv) and schematic Block Design Files (.bdf). In addition, you can combine your logic design files with Intel and third-party IP core design files, including combining components into a Platform Designer system (.qsys).

The New Project Wizard prompts you to identify logic design files. Add or remove project files by clicking **Project > Add/Remove Files in Project**. View the project's logic design files in the Project Navigator.

**Figure 12. Design and IP Files in Project Navigator**



Right-click files in the Project Navigator to:

- **Open** and edit the file
- **Remove File from Project**
- **Set as Top-Level Entity** for the project revision
- **Create a Symbol File for Current File** for display in schematic editors
- Edit file **Properties**

### 2.6.1 Including Design Libraries

Include design files libraries in your project. Specify libraries for a single project, or for all Intel Quartus Prime projects. The .qsf stores project library information.

The quartus2.ini file stores global library information.

#### Related Links

[Design Library Migration Guidelines](#) on page 80



### 2.6.1.1 Specifying Design Libraries

Follow these steps to specify project libraries from the GUI.

1. Click **Assignment > Settings**.
2. Click **Libraries** and specify the **Project Library name** or **Global Library name**. Alternatively, you can specify project libraries with `SEARCH_PATH` in the `.qsf`, and global libraries in the `quartus2.ini` file.

## 2.7 Managing Timing Constraints

Apply appropriate timing constraints to correctly optimize fitting and analyze timing for your design. The Fitter optimizes the placement of logic in the device to meet your specified timing and routing constraints.

Specify timing constraints in the Timing Analyzer (**Tools > Timing Analyzer**), or in an `.sdc` file. Specify constraints for clock characteristics, timing exceptions, and external signal setup and hold times before running analysis. Timing Analyzer reports the detailed information about the performance of your design compared with constraints in the Compilation Report panel.

Save the constraints you specify in the GUI in an industry-standard Synopsys Design Constraints File (`.sdc`). You can subsequently edit the text-based `.sdc` file directly. If you refer to multiple `.sdc` files in a parent `.sdc` file, the Timing Analyzer reads the `.sdc` files in the order you list.

## 2.8 Introduction to Intel FPGA IP Cores

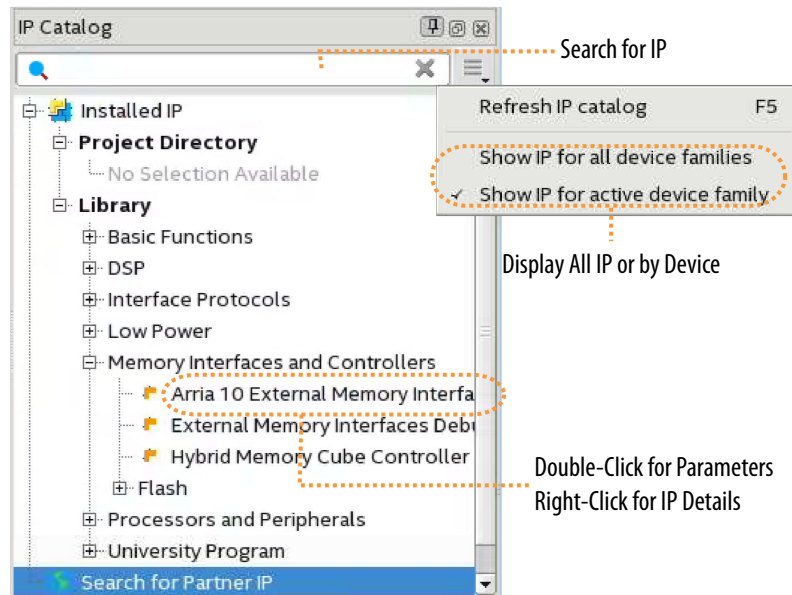
Intel and strategic IP partners offer a broad portfolio of configurable IP cores optimized for Intel FPGA devices.

The Intel Quartus Prime software installation includes the Intel FPGA IP library. Integrate optimized and verified Intel FPGA IP cores into your design to shorten design cycles and maximize performance. The Intel Quartus Prime software also supports integration of IP cores from other sources. Use the IP Catalog (**Tools > IP Catalog**) to efficiently parameterize and generate synthesis and simulation files for your custom IP variation. The Intel FPGA IP library includes the following types of IP cores:

- Basic functions
- DSP functions
- Interface protocols
- Low power functions
- Memory interfaces and controllers
- Processors and peripherals

This document provides basic information about parameterizing, generating, upgrading, and simulating stand-alone IP cores in the Intel Quartus Prime software.

Figure 13. IP Catalog



### 2.8.1 IP Catalog and Parameter Editor

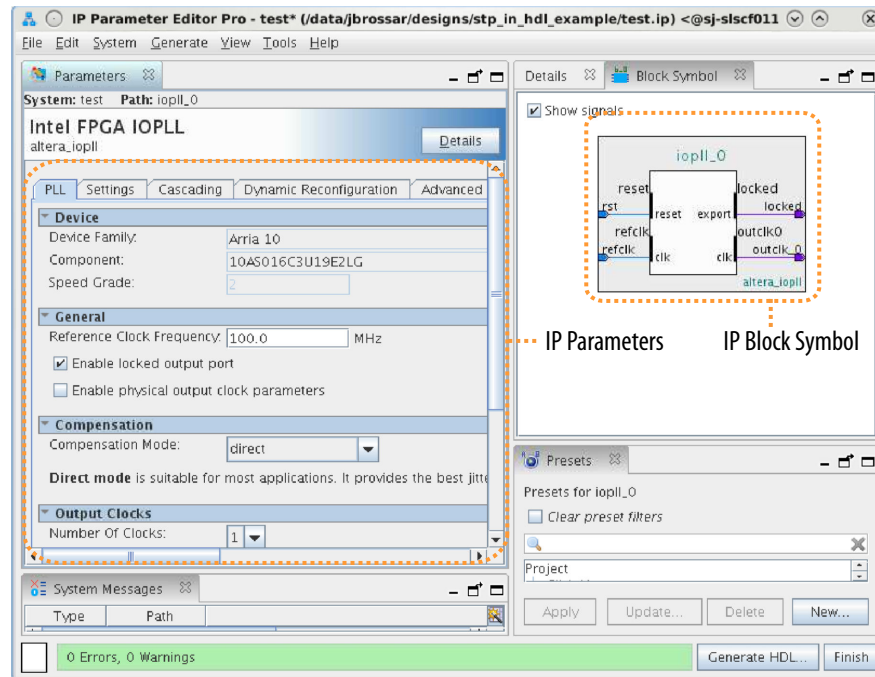
The IP Catalog displays the IP cores available for your project. Use the following features of the IP Catalog to locate and customize an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, to open the IP core's installation folder, and for links to IP documentation.
- Click **Search for Partner IP** to access partner IP information on the web.

The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Intel Quartus Prime IP file (.ip) for an IP variation in Intel Quartus Prime Pro Edition projects.



Figure 14. IP Parameter Editor (Intel Quartus Prime Pro Edition)



### Related Links

[Creating a System with Platform Designer](#) on page 327

#### 2.8.1.1 The Parameter Editor

The parameter editor helps you to configure IP core ports, parameters, and output file generation options. The basic parameter editor controls include the following:

- Use the **Presets** window to apply preset parameter values for specific applications (for select cores).
- Use the **Details** window to view port and parameter descriptions, and click links to documentation.
- Click **Generate > Generate Testbench System** to generate a testbench system (for select cores).
- Click **Generate > Generate Example Design** to generate an example design (for select cores).
- Click **Validate System Integrity** to validate a system's generic components against companion files. (Platform Designer systems only)
- Click **Sync All System Info** to validate a system's generic components against companion files. (Platform Designer systems only)

The IP Catalog is also available in Platform Designer (**View > IP Catalog**). The Platform Designer IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Intel Quartus Prime IP Catalog. Refer to *Creating a System with Platform Designer* or *Creating a System with Platform Designer* for information on use of IP in Platform Designer and Platform Designer, respectively.

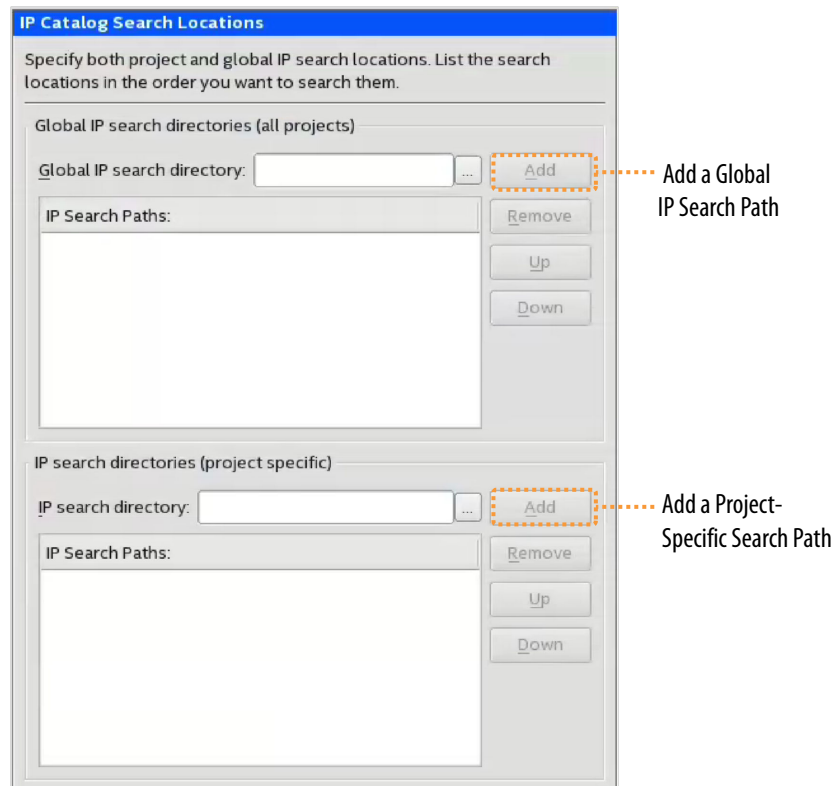
### Related Links

- [Creating a System with Platform Designer](#)
- [Creating a System with Platform Designer](#)

### 2.8.1.2 Adding IP Cores to IP Catalog

The IP Catalog automatically displays IP cores located in the project directory, in the default Intel Quartus Prime installation directory, and in the IP search path.

**Figure 15. Specifying IP Search Locations**



The IP Catalog displays Intel Quartus Prime IP components and Platform Designer systems, third-party IP components, and any custom IP components that you include in the path. Use the **IP Search Path** option (**Tools > Options**) to include custom and third-party IP components in the IP Catalog.

The Intel Quartus Prime software searches the directories listed in the IP search path for the following IP core files:

- Component Description File (`_hw.tcl`)—defines a single IP core.
- IP Index File (`.ipx`)—each `.ipx` file indexes a collection of available IP cores. This file specifies the relative path of directories to search for IP cores. In general, `.ipx` files facilitate faster searches.

The Intel Quartus Prime software searches some directories recursively and other directories only to a specific depth. When the search is recursive, the search stops at any directory that contains a `_hw.tcl` or `.ipx` file.





In the following list of search locations, \*\* indicates a recursive descent.

**Table 16. IP Search Locations**

Location	Description
PROJECT_DIR/*	Finds IP components and index files in the Intel Quartus Prime project directory.
PROJECT_DIR/ip/**/*	Finds IP components and index files in any subdirectory of the /ip subdirectory of the Intel Quartus Prime project directory.

If the Intel Quartus Prime software recognizes two IP cores with the same name, the following search path precedence rules determine the resolution of files:

1. Project directory.
2. Project database directory.
3. Project IP search path specified in **IP Search Locations**, or with the SEARCH\_PATH assignment for the current project revision.
4. Global IP search path specified in **IP Search Locations**, or with the SEARCH\_PATH assignment in the quartus2.ini file.
5. Quartus software libraries directory, such as *<Quartus Installation>\libraries*.

**Note:** If you add an IP component to the search path, update the IP Catalog by clicking **Refresh IP Catalog** in the drop-down list. In Platform Designer and Platform Designer, click **File > Refresh System** to update the IP Catalog.

### 2.8.1.3 General Settings for IP

Use the following settings to control how the Intel Quartus Prime software manages IP cores in your project.

**Table 17. IP Core General Setting Locations**

Setting Location	Description
<b>Tools &gt; Options &gt; IP Settings</b> Or <b>Tasks pane &gt; Settings &gt; IP Settings</b> (Pro Edition Only)	<ul style="list-style-type: none"> <li>• Specify the <b>IP generation HDL preference</b>. The parameter editor generates the HDL you specify for IP variations.</li> <li>• Increase the <b>Maximum Platform Designer memory usage size</b> if you experience slow processing for large systems, or for out of memory errors.</li> <li>• Specify whether to <b>Automatically add Intel Quartus Prime IP files</b> to all projects. Disable this option to manually add the IP files.</li> <li>• Use the <b>IP Regeneration Policy</b> setting to control when synthesis files regenerate for each IP variation. Typically, you <b>Always regenerate synthesis files for IP cores</b> after making changes to an IP variation.</li> </ul>
<b>Tools &gt; Options &gt; IP Catalog Search Locations</b> Or <b>Tasks pane &gt; Settings &gt; IP Catalog Search Locations</b> (Pro Edition Only)	<ul style="list-style-type: none"> <li>• Specify additional project and global IP search locations. The Intel Quartus Prime software searches for IP cores in the project directory, in the Intel Quartus Prime installation directory, and in the IP search path.</li> </ul>

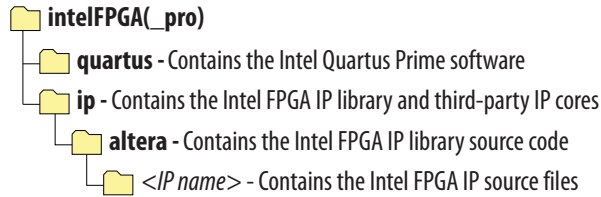
### 2.8.1.4 Installing and Licensing Intel FPGA IP Cores

The Intel Quartus Prime software installation includes the Intel FPGA IP library. This library provides many useful IP cores for your production use without the need for an additional license. Some Intel FPGA IP cores require purchase of a separate license for

production use. The Intel FPGA IP Evaluation Mode allows you to evaluate these licensed Intel FPGA IP cores in simulation and hardware, before deciding to purchase a full production IP core license. You only need to purchase a full production license for licensed Intel IP cores after you complete hardware testing and are ready to use the IP in production.

The Intel Quartus Prime software installs IP cores in the following locations by default:

**Figure 16. IP Core Installation Path**



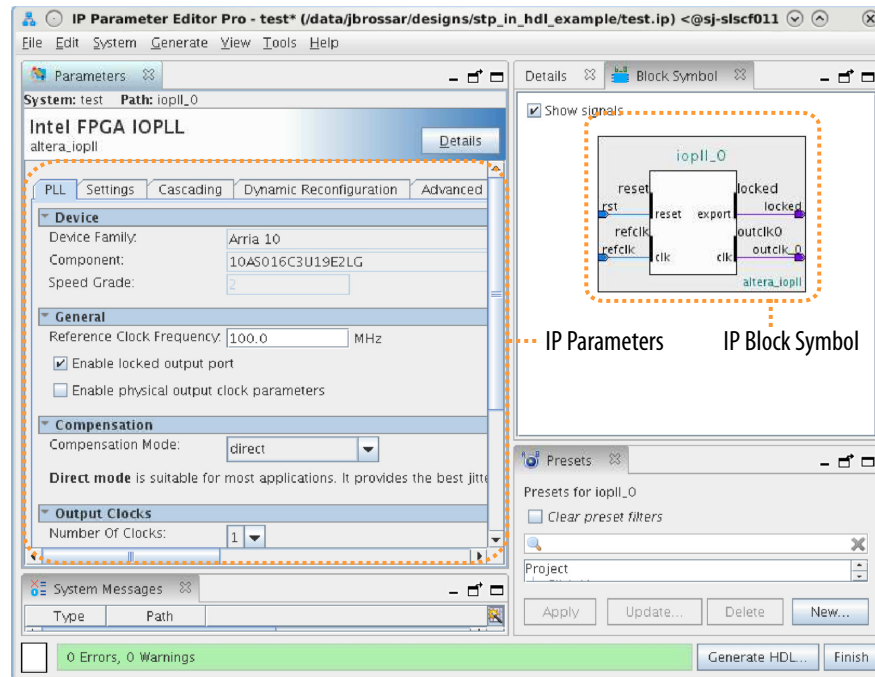
**Table 18. IP Core Installation Locations**

Location	Software	Platform
<drive>:\intelFPGA_pro\quartus\ip\altera	Intel Quartus Prime Pro Edition	Windows*
<drive>:\intelFPGA\quartus\ip\altera	Intel Quartus Prime Standard Edition	Windows
<home directory>:/intelFPGA_pro/quartus/ip/altera	Intel Quartus Prime Pro Edition	Linux*
<home directory>:/intelFPGA/quartus/ip/altera	Intel Quartus Prime Standard Edition	Linux

### 2.8.2 Generating IP Cores (Intel Quartus Prime Pro Edition)

Quickly configure Intel FPGA IP cores in the Intel Quartus Prime parameter editor. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to define a custom variation of the IP core. The parameter editor generates the IP variation synthesis and optional simulation files, and adds the .ip file representing the variation to your project automatically.

Figure 17. IP Parameter Editor (Intel Quartus Prime Pro Edition)



Follow these steps to locate, instantiate, and customize an IP core in the parameter editor:

1. Create or open an Intel Quartus Prime project (.qpf) to contain the instantiated IP variation.
  2. In the IP Catalog (**Tools > IP Catalog**), locate and double-click the name of the IP core to customize. To locate a specific component, type some or all of the component's name in the IP Catalog search box. The New IP Variation window appears.
  3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file named *<your\_ip>.ip*. Click **OK**. The parameter editor appears.
  4. Set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in IP parameters:
    - Optionally, select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
    - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
    - Specify options for processing the IP core files in other EDA tools.
- Note:* Refer to your IP core user guide for information about specific IP core parameters.
5. Click **Generate HDL**. The **Generation** dialog box appears.
  6. Specify output file generation options, and then click **Generate**. The synthesis and simulation files generate according to your specifications.



7. To generate a simulation testbench, click **Generate** ► **Generate Testbench System**. Specify testbench generation options, and then click **Generate**.
8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate** ► **Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project.
10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

*Note:* Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

### 2.8.2.1 IP Core Generation Output (Intel Quartus Prime Pro Edition)

The Intel Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.



Figure 18. Individual IP Core Generation Output (Intel Quartus Prime Pro Edition)

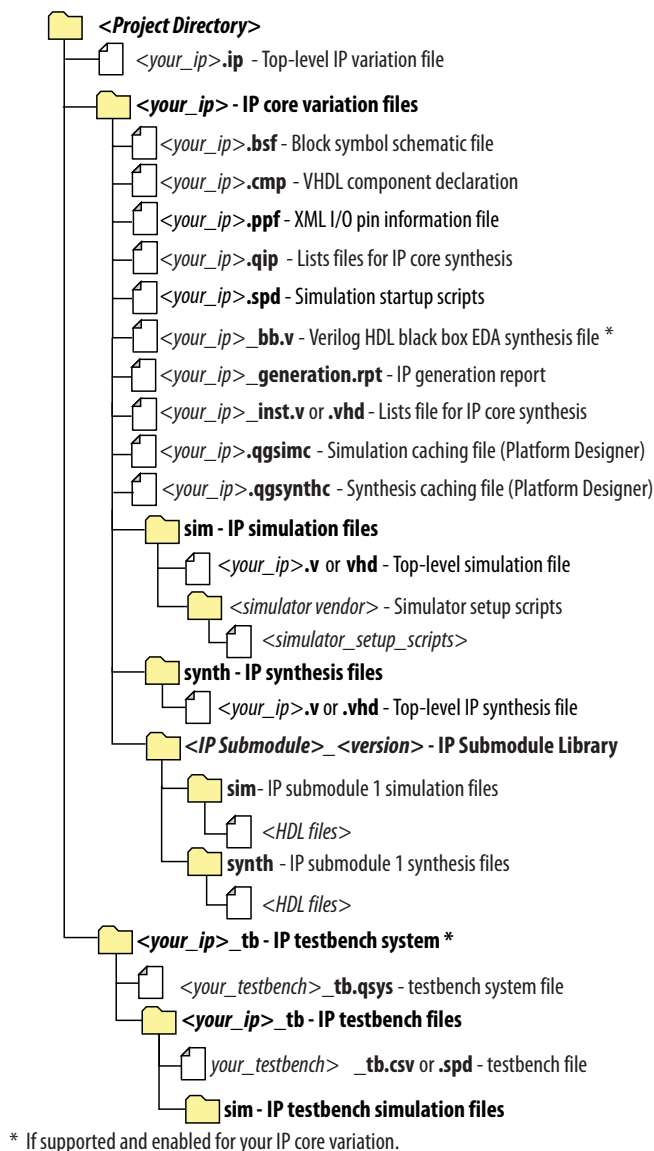


Table 19. Output Files of Intel FPGA IP Generation

File Name	Description
<your_ip>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a .qsys file.
<your_ip>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<your_ip>_generation.rpt	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.

*continued...*



File Name	Description
<your_ip>.qgsimc (Platform Designer systems only)	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<your_ip>.qgsynth (Platform Designer systems only)	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<your_ip>.qip	Contains all information to integrate and compile the IP component.
<your_ip>.csv	Contains information about the upgrade status of the IP component.
<your_ip>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (.bdf).
<your_ip>.spd	Input file that ip-make-simscript requires to generate simulation scripts. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.
<your_ip>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.
<your_ip>_bb.v	Use the Verilog blackbox (_bb.v) file as an empty module declaration for use as a blackbox.
<your_ip>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<your_ip>.regmap	If the IP contains register information, the Intel Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<your_ip>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Intel Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system slaves, Platform Designer accesses the registers by name.
<your_ip>.v <your_ip>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a msim_setup.tcl script to set up and run a ModelSim simulation.
aldec/	Contains a Riviera*-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file to set up and run a VCS MX* simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules	Contains HDL files for the IP core submodule.
<IP submodule>/	Platform Designer generates /synth and /sim sub-directories for each IP submodule directory that Platform Designer generates.

### 2.8.2.2 Scripting IP Core Generation

Use the qsys-script and qsys-generate utilities to define and generate an IP core variation outside of the Intel Quartus Prime GUI.



To parameterize and generate an IP core at command-line, follow these steps:

1. Run `qsys-script` to start a Tcl script that instantiates the IP and sets desired parameters:

```
qsys-script --script=<script_file>.tcl
```

2. Run `qsys-generate` to generate the IP core variation:

```
qsys-generate <IP variation file>.qsys
```

**Table 20. qsys-generate Command-Line Options**

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the .qsys system file to generate.
--synthesis=<VERILOG VHDL>	Optional	Creates synthesis HDL files that Platform Designer uses to compile the system in an Intel Quartus Prime project. Specify the generation language for the top-level RTL file for the Platform Designer system. The default value is <i>VERILOG</i> .
--block-symbol-file	Optional	Creates a Block Symbol File (.bsf) for the Platform Designer system.
--greybox	Optional	If you are synthesizing your design with a third-party EDA synthesis tool, generate a netlist for the synthesis tool to estimate timing and resource usage for this design.
--ipxact	Optional	If you set this option to true, Platform Designer renders the post-generation system as an IPXACT-compatible component description.
--simulation=<VERILOG VHDL>	Optional	Creates a simulation model for the Platform Designer system. The simulation model contains generated HDL files for the simulator, and may include simulation-only features. Specify the preferred simulation language. The default value is <i>VERILOG</i> .
--testbench=<SIMPLE STANDARD>	Optional	Creates a testbench system that instantiates the original system, adding bus functional models (BFMs) to drive the top-level interfaces. When you generate the system, the BFMs interact with the system in the simulator. The default value is <i>STANDARD</i> .
--testbench-simulation=<VERILOG VHDL>	Optional	After you create the testbench system, create a simulation model for the testbench system. The default value is <i>VERILOG</i> .
--example-design=<value>	Optional	Creates example design files. For example, <code>--example-design</code> or <code>--example-design=all</code> . The default is <i>All</i> , which generates example designs for all instances. Alternatively, choose specific filesets based on instance name and fileset name. For example <code>--example-design=instance0.example_design1,instance1.example_design 2</code> . Specify an output directory for the example design files creation.
--search-path=<value>	Optional	If you omit this command, Platform Designer uses a standard default path. If you provide this command, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example, <code>"/extra/dir,\$"</code> .
--family=<value>	Optional	Sets the device family name.

*continued...*



Option	Usage	Description
<code>--part=&lt;value&gt;</code>	Optional	Sets the device part number. If set, this option overrides the <code>--family</code> option.
<code>--upgrade-variation-file</code>	Optional	If you set this option to true, the file argument for this command accepts a <code>.v</code> file, which contains a IP variant. This file parameterizes a corresponding instance in a Platform Designer system of the same name.
<code>--upgrade-ip-cores</code>	Optional	Enables upgrading all the IP cores that support upgrade in the Platform Designer system.
<code>--clear-output-directory</code>	Optional	Clears the output directory corresponding to the selected target, that is, simulation or synthesis.
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	The maximum memory size that Platform Designer uses when running <code>qsys-generate</code> . You specify the value as <code>&lt;size&gt;&lt;unit&gt;</code> , where <code>unit</code> is <code>m</code> (or <code>M</code> ) for multiples of megabytes or <code>g</code> (or <code>G</code> ) for multiples of gigabytes. The default value is 512m.
<code>--help</code>	Optional	Displays help for <code>--qsys-generate</code> .

### 2.8.3 Modifying an IP Variation

After generating an IP core variation, use any of the following methods to modify the IP variation in the parameter editor.

**Table 21. Modifying an IP Variation**

Menu Command	Action
<b>File &gt; Open</b>	Select the top-level HDL ( <code>.v</code> , or <code>.vhd</code> ) IP variation file to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes.
<b>View &gt; Project Navigator &gt; IP Components</b>	Double-click the IP variation to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes.
<b>Project &gt; Upgrade IP Components</b>	Select the IP variation and click <b>Upgrade in Editor</b> to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes.

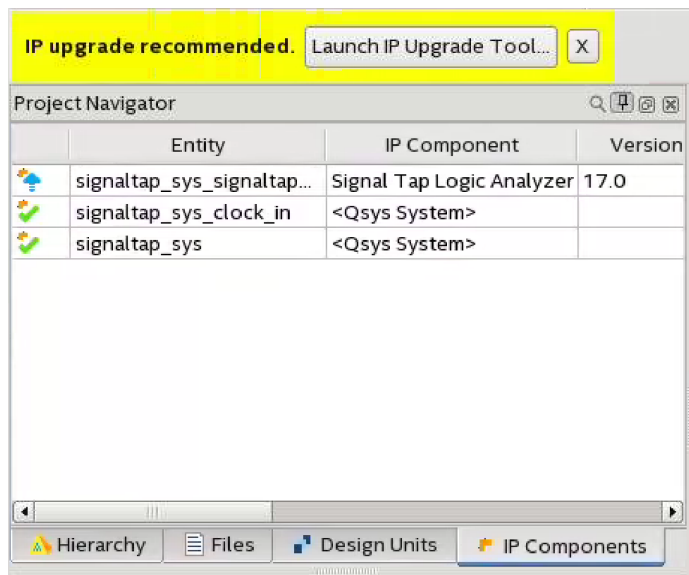
### 2.8.4 Upgrading IP Cores

Any Intel FPGA IP variations that you generate from a previous version or different edition of the Intel Quartus Prime software, may require upgrade before compilation in the current software edition or version. The Project Navigator displays a banner indicating the IP upgrade status. Click **Launch IP Upgrade Tool** or **Project > Upgrade IP Components** to upgrade outdated IP cores.





Figure 19. IP Upgrade Alert in Project Navigator






Icons in the **Upgrade IP Components** dialog box indicate when IP upgrade is required, optional, or unsupported for an IP variation in the project. Upgrade IP variations that require upgrade before compilation in the current version of the Intel Quartus Prime software.

**Note:** Upgrading IP cores may append a unique identifier to the original IP core entity names, without similarly modifying the IP instance name. There is no requirement to update these entity references in any supporting Intel Quartus Prime file, such as the Intel Quartus Prime Settings File (.qsf), Synopsys\* Design Constraints File (.sdc), or Signal Tap File (.stp), if these files contain instance names. The Intel Quartus Prime software reads only the instance name and ignores the entity name in paths that specify both names. Use only instance names in assignments.

Table 22. IP Core Upgrade Status

IP Core Status	Description
 IP Upgraded	Indicates that your IP variation uses the latest version of the Intel FPGA IP core.
 IP Component Outdated	Indicates that your IP variation uses an outdated version of the IP core.
 IP End of Life	Indicates that Intel designates the IP core as end-of-life status. You may or may not be able to edit the IP core in the parameter editor. Support for this IP core discontinues in future releases of the Intel Quartus Prime software.

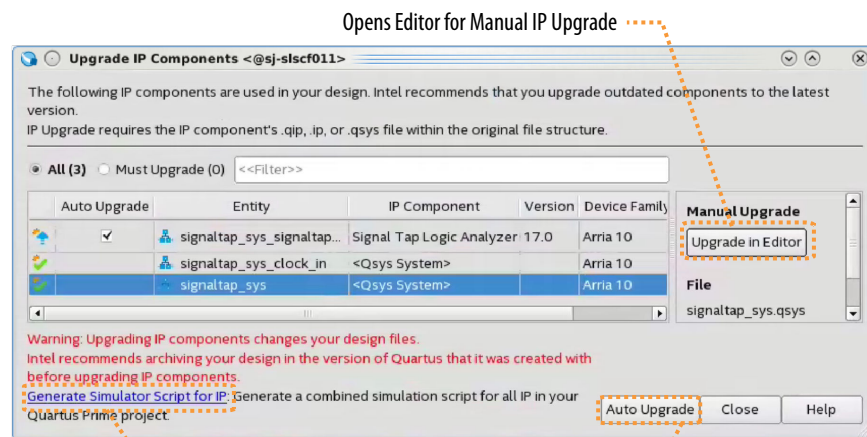
*continued...*

IP Core Status	Description
IP Upgrade Mismatch Warning 	Provides warning of non-critical IP core differences in migrating IP to another device family.
IP has incompatible subcores 	Indicates that the current version of the Intel Quartus Prime software does not support compilation of your IP variation, because the IP has incompatible subcores
Compilation of IP Not Supported 	Indicates that the current version of the Intel Quartus Prime software does not support compilation of your IP variation. This can occur if another edition of the Intel Quartus Prime software, such as the Intel Quartus Prime Standard Edition, generated this IP. Replace this IP component with a compatible component in the current edition.

Follow these steps to upgrade IP cores:

1. In the latest version of the Intel Quartus Prime software, open the Intel Quartus Prime project containing an outdated IP core variation. The **Upgrade IP Components** dialog box automatically displays the status of IP cores in your project, along with instructions for upgrading each core. To access this dialog box manually, click **Project > Upgrade IP Components**.
2. To upgrade one or more IP cores that support automatic upgrade, ensure that you turn on the **Auto Upgrade** option for the IP cores, and click **Auto Upgrade**. The **Status** and **Version** columns update when upgrade is complete. Example designs that any Intel FPGA IP core provides regenerate automatically whenever you upgrade an IP core.
3. To manually upgrade an individual IP core, select the IP core and click **Upgrade in Editor** (or simply double-click the IP core name). The parameter editor opens, allowing you to adjust parameters and regenerate the latest version of the IP core.

Figure 20. Upgrading IP Cores



Opens Editor for Manual IP Upgrade

Runs "Auto Upgrade" on all Outdated Cores

Generates/Updates Combined Simulation Setup Script for all Project IP



*Note:* Intel FPGA IP cores older than Intel Quartus Prime software version 12.0 do not support upgrade. Intel verifies that the current version of the Intel Quartus Prime software compiles the previous two versions of each IP core. The *Intel FPGA IP Core Release Notes* reports any verification exceptions for Intel FPGA IP cores. Intel does not verify compilation for IP cores older than the previous two releases.

### Related Links

[Intel FPGA IP Core Release Notes](#)

## 2.8.4.1 Upgrading IP Cores at Command-Line

Optionally, upgrade an Intel FPGA IP core at the command-line, rather than using the GUI. IP cores that do not support automatic upgrade do not support command-line upgrade.

- To upgrade a single IP core at the command-line, type the following command:

```
quartus_sh -ip_upgrade -variation_files <my_ip>.<qsys,.v, .vhd>  
<quartus_project>
```

Example:

```
quartus_sh -ip_upgrade -variation_files mega/pll25.qsys hps_testx
```

- To simultaneously upgrade multiple IP cores at the command-line, type the following command:

```
quartus_sh -ip_upgrade -variation_files "<my_ip1>.<qsys,.v, .vhd>>;  
<my_ip_filepath/my_ip2>.<hdl>" <quartus_project>
```

Example:

```
quartus_sh -ip_upgrade -variation_files "mega/pll_tx2.qsys;mega/  
pll3.qsys" hps_testx
```

## 2.8.4.2 Migrating IP Cores to a Different Device

Migrate an Intel FPGA IP variation when you want to target a different (often newer) device. Most Intel FPGA IP cores support automatic migration. Some IP cores require manual IP regeneration for migration. A few IP cores do not support device migration, requiring you to replace them in the project. The **Upgrade IP Components** dialog box identifies the migration support level for each IP core in the design.



1. To display the IP cores that require migration, click **Project > Upgrade IP Components**. The **Description** field provides migration instructions and version differences.
2. To migrate one or more IP cores that support automatic upgrade, ensure that the **Auto Upgrade** option is turned on for the IP cores, and click **Perform Automatic Upgrade**. The **Status** and **Version** columns update when upgrade is complete.
3. To migrate an IP core that does not support automatic upgrade, double-click the IP core name, and click **OK**. The parameter editor appears. If the parameter editor specifies a **Currently selected device family**, turn off **Match project/default**, and then select the new target device family.
4. Click **Generate HDL**, and confirm the **Synthesis** and **Simulation** file options. Verilog HDL is the default output file format. If you specify VHDL as the output format, select **VHDL** to retain the original output format.
5. Click **Finish** to complete migration of the IP core. Click **OK** if the software prompts you to overwrite IP core files. The **Device Family** column displays the new target device name when migration is complete.
6. To ensure correctness, review the latest parameters in the parameter editor or generated HDL.

*Note:* IP migration may change ports, parameters, or functionality of the IP variation. These changes may require you to modify your design or to re-parameterize your IP variant. During migration, the IP variation's HDL generates into a library that is different from the original output location of the IP core. Update any assignments that reference outdated locations. If a symbol in a supporting Block Design File schematic represents your upgraded IP core, replace the symbol with the newly generated `<my_ip>.bsf`. Migration of some IP cores requires installed support for the original and migration device families.

### Related Links

[Intel FPGA IP Release Notes](#)

#### 2.8.4.3 Troubleshooting IP or Platform Designer System Upgrade

The **Upgrade IP Components** dialog box reports the version and status of each IP core and Platform Designer system following upgrade or migration.

If any upgrade or migration fails, the **Upgrade IP Components** dialog box provides information to help you resolve any errors.

*Note:* Do not use spaces in IP variation names or paths.

During automatic or manual upgrade, the Messages window dynamically displays upgrade information for each IP core or Platform Designer system. Use the following information to resolve upgrade errors:



**Table 23. IP Upgrade Error Information**

Upgrade IP Components Field	Description
<b>Status</b>	Displays the "Success" or "Failed" status of each upgrade or migration. Click the status of any upgrade that fails to open the <b>IP Upgrade Report</b> .
<b>Version</b>	Dynamically updates the version number when upgrade is successful. The text is red when the IP requires upgrade.
<b>Device Family</b>	Dynamically updates to the new device family when migration is successful. The text is red when the IP core requires upgrade.
<b>Auto Upgrade</b>	Runs automatic upgrade on all IP cores that support auto upgrade. Also, automatically generates a <code>&lt;Project Directory&gt;/ip_upgrade_port_diff_report</code> report for IP cores or Platform Designer systems that fail upgrade. Review these reports to determine any port differences between the current and previous IP core version.

Use the following techniques to resolve errors if your IP core or Platform Designer system "Failed" to upgrade versions or migrate to another device. Review and implement the instructions in the **Description** field, including one or more of the following:

- If the current version of the software does not support the IP variant, right-click the component and click **Remove IP Component from Project**. Replace this IP core or Platform Designer system with the one supported in the current version of the software.
- If the current target device does not support the IP variant, select a supported device family for the project, or replace the IP variant with a suitable replacement that supports your target device.
- If an upgrade or migration fails, click **Failed** in the **Status** field to display and review details of the **IP Upgrade Report**. Click the **Release Notes** link for the latest known issues about the IP core. Use this information to determine the nature of the upgrade or migration failure and make corrections before upgrade.
- Run **Auto Upgrade** to automatically generate an **IP Ports Diff** report for each IP core or Platform Designer system that fails upgrade. Review the reports to determine any port differences between the current and previous IP core version. Click **Upgrade in Editor** to make specific port changes and regenerate your IP core or Platform Designer system.
- If your IP core or Platform Designer system does not support **Auto Upgrade**, click **Upgrade in Editor** to resolve errors and regenerate the component in the parameter editor.

Figure 21. IP Upgrade Report



### 2.8.5 Simulating Intel FPGA IP Cores

The Intel Quartus Prime software supports IP core RTL simulation in specific EDA simulators. IP generation creates simulation files, including the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts for each IP core. Use the functional simulation model and any testbench or example design for simulation. IP generation output may also include scripts to compile and run any testbench. The scripts list all models or libraries you require to simulate your IP core.

The Intel Quartus Prime software provides integration with many simulators and supports multiple simulation flows, including your own scripted and custom simulation flows. Whichever flow you choose, IP core simulation involves the following steps:

1. Generate simulation model, testbench (or example design), and simulator setup script files.
2. Set up your simulator environment and any simulation scripts.
3. Compile simulation model libraries.
4. Run your simulator.



### 2.8.5.1 Generating IP Simulation Files

The Intel Quartus Prime software optionally generates the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts when you generate an IP core. To control the generation of IP simulation files:

- To specify your supported simulator and options for IP simulation file generation, click **Assignment > Settings > EDA Tool Settings > Simulation**.
- To parameterize a new IP variation, enable generation of simulation files, and generate the IP core synthesis and simulation files, click **Tools > IP Catalog**.
- To edit parameters and regenerate synthesis or simulation files for an existing IP core variation, click **View > Project Navigator > IP Components**.

**Table 24. Intel FPGA IP Simulation Files**

File Type	Description	File Name
Simulator setup scripts	Vendor-specific scripts to compile, elaborate, and simulate Intel FPGA IP models and simulation model library files. Optionally, generate a simulator setup script for each vendor that combines the individual IP core scripts into one file. Source the combined script from your top-level simulation script to eliminate script maintenance.	<code>&lt;my_dir&gt;/aldec/ rivierapro_setup.tcl</code> <code>&lt;my_dir&gt;/cadence/ ncsim_setup.sh</code> <code>&lt;my_dir&gt;/mentor/msim_setup.tcl</code> <code>&lt;my_dir&gt;/synopsys/vcs/ vcs_setup.sh</code>

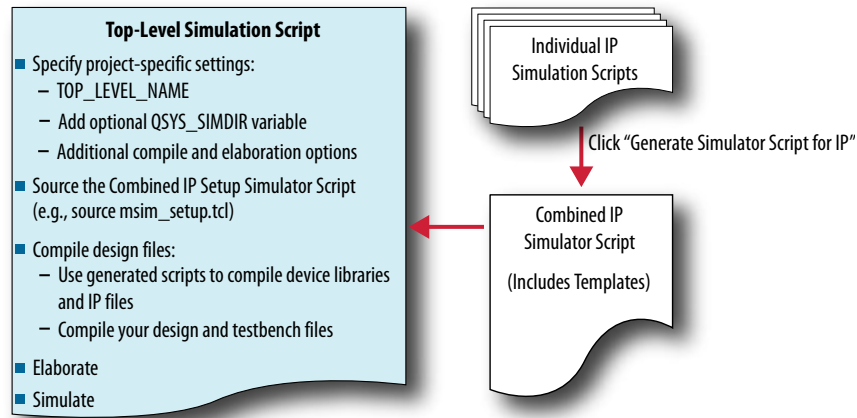
**Note:** Intel FPGA IP cores support a variety of cycle-accurate simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. The models support fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some IP cores, generation only produces the plain text RTL model, and you can simulate that model. Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

### 2.8.5.2 Scripting IP Simulation

The Intel Quartus Prime software supports the use of scripts to automate simulation processing in your preferred simulation environment. Use the scripting methodology that you prefer to control simulation.

Use a version-independent, top-level simulation script to control design, testbench, and IP core simulation. Because Intel Quartus Prime-generated simulation file names may change after IP upgrade or regeneration, your top-level simulation script must "source" the generated setup scripts, rather than using the generated setup scripts directly. Follow these steps to generate or regenerate combined simulator setup scripts:

**Figure 22. Incorporating Generated Simulator Setup Scripts into a Top-Level Simulation Script**



1. Click **Project > Upgrade IP Components > Generate Simulator Script for IP** (or run the `ip-setup-simulation` utility) to generate or regenerate a combined simulator setup script for all IP for each simulator.
2. Use the templates in the generated script to source the combined script in your top-level simulation script. Each simulator's combined script file contains a rudimentary template that you adapt for integration of the setup script into a top-level simulation script.

This technique eliminates manual update of simulation scripts if you modify or upgrade the IP variation.

#### 2.8.5.2.1 Generating a Combined Simulator Setup Script

Run the **Generate Simulator Setup Script for IP** command to generate a combined simulator setup script.

Source this combined script from a top-level simulation script. Click **Tools > Generate Simulator Setup Script for IP** (or use of the `ip-setup-simulation` utility at the command-line) to generate or update the combined scripts, after any of the following occur:

- IP core initial generation or regeneration with new parameters
- Intel Quartus Prime software version upgrade
- IP core version upgrade

To generate a combined simulator setup script for all project IP cores for each simulator:





1. Generate, regenerate, or upgrade one or more IP core. Refer to *Generating IP Cores* or *Upgrading IP Cores*.
2. Click **Tools** ► **Generate Simulator Setup Script for IP** (or run the `ip-setup-simulation` utility). Specify the **Output Directory** and library compilation options. Click **OK** to generate the file. By default, the files generate into the `/ <project directory>/<simulator>/` directory using relative paths.
3. To incorporate the generated simulator setup script into your top-level simulation script, refer to the template section in the generated simulator setup script as a guide to creating a top-level script:
  - a. Copy the specified template sections from the simulator-specific generated scripts and paste them into a new top-level file.
  - b. Remove the comments at the beginning of each line from the copied template sections.
  - c. Specify the customizations you require to match your design simulation requirements, for example:
    - Specify the `TOP_LEVEL_NAME` variable to the design's simulation top-level file. The top-level entity of your simulation is often a testbench that instantiates your design. Then, your design instantiates IP cores or Platform Designer systems. Set the value of `TOP_LEVEL_NAME` to the top-level entity.
    - If necessary, set the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files.
    - Compile the top-level HDL file (for example, a test program) and all other files in the design.
    - Specify any other changes, such as using the `grep` command-line utility to search a transcript file for error signatures, or e-mail a report.
4. Re-run **Tools** ► **Generate Simulator Setup Script for IP** (or `ip-setup-simulation`) after regeneration of an IP variation.

**Table 25. Simulation Script Utilities**

Utility	Syntax
<p><code>ip-setup-simulation</code> generates a combined, version-independent simulation script for all Intel FPGA IP cores in your project. The command also automates regeneration of the script after upgrading software or IP versions. Use the <code>compile-to-work</code> option to compile all simulation files into a single work library if your simulation environment requires. Use the <code>--use-relative-paths</code> option to use relative paths whenever possible.</p>	<pre>ip-setup-simulation --quartus-project=&lt;my proj&gt; --output-directory=&lt;my_dir&gt; --use-relative-paths --compile-to-work</pre> <p><code>--use-relative-paths</code> and <code>--compile-to-work</code> are optional. For command-line help listing all options for these executables, type: <code>&lt;utility name&gt; --help</code>.</p>
<p><code>ip-make-simscript</code> generates a combined simulation script for all IP cores that you specify on the command line. Specify one or more <code>.spd</code> files and an output directory in the command. Running the script compiles IP simulation models into various simulation libraries.</p>	<pre>ip-make-simscript --spd=&lt;ipA.spd, ipB.spd&gt; --output-directory=&lt;directory&gt;</pre>

The following sections provide step-by-step instructions for sourcing each simulator setup script in your top-level simulation script.



## Sourcing Aldec\* Simulator Setup Scripts

Follow these steps to incorporate the generated Aldec simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.tcl`.

```
# # Start of template
# # If the copied and modified template file is "aldec.do", run it as:
# # vsim -c -do aldec.do
# #
# # Source the generated sim script
# source rivierapro_setup.tcl
# # Compile eda/sim_lib contents first
# dev_com
# # Override the top-level name (so that elab is useful)
# set TOP_LEVEL_NAME top
# # Compile the standalone IP.
# com
# # Compile the top-level
# vlog -sv2k5 ../../top.sv
# # Elaborate the design.
# elab
# # Run the simulation
# run
# # Report success to the shell
# exit -code 0
# # End of template
```

2. Delete the first two characters of each line (comment and space):

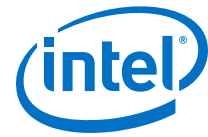
```
# Start of template
# If the copied and modified template file is "aldec.do", run it as:
# vsim -c -do aldec.do
#
# Source the generated sim script source rivierapro_setup.tcl
# Compile eda/sim_lib contents first dev_com
# Override the top-level name (so that elab is useful)
set TOP_LEVEL_NAME top
# Compile the standalone IP.
com
# Compile the top-level vlog -sv2k5 ../../top.sv
# Elaborate the design.
elab
# Run the simulation
run
# Report success to the shell
exit -code 0
# End of template
```

3. Modify the `TOP_LEVEL_NAME` and compilation step appropriately, depending on the simulation's top-level file. For example:

```
set TOP_LEVEL_NAME sim_top
vlog -sv2k5 ../../sim_top.sv
```

4. If necessary, add the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the new top-level script from the generated simulation directory:

```
vsim -c -do <path to sim_top>.tcl
```



## Sourcing Cadence\* Simulator Setup Scripts

Follow these steps to incorporate the generated Cadence IP simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `ncsim.sh`.

```
# # Start of template
# # If the copied and modified template file is "ncsim.sh", run it as:
# # ./ncsim.sh
# #
# # Do the file copy, dev_com and com steps
# source ncsim_setup.sh \
# SKIP_ELAB=1 \
# SKIP_SIM=1
#
# # Compile the top level module
# ncvlog -sv "$QSYS_SIMDIR/./top.sv"
#
# # Do the elaboration and sim steps
# # Override the top-level name
# # Override the sim options, so the simulation
# # runs forever (until $finish()).
# source ncsim_setup.sh \
# SKIP_FILE_COPY=1 \
# SKIP_DEV_COM=1 \
# SKIP_COM=1 \
# TOP_LEVEL_NAME=top \
# USER_DEFINED_SIM_OPTIONS=""
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "ncsim.sh", run it as:
# ./ncsim.sh
#
# Do the file copy, dev_com and com steps
source ncsim_setup.sh \
SKIP_ELAB=1 \
SKIP_SIM=1
# Compile the top level module
ncvlog -sv "$QSYS_SIMDIR/./top.sv"
# Do the elaboration and sim steps
# Override the top-level name
# Override the sim options, so the simulation
# runs forever (until $finish()).
source ncsim_setup.sh \
SKIP_FILE_COPY=1 \
SKIP_DEV_COM=1 \
SKIP_COM=1 \
```



```
TOP_LEVEL_NAME=top \  
USER_DEFINED_SIM_OPTIONS=""  
# End of template
```

3. Modify the `TOP_LEVEL_NAME` and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=sim_top \  
ncvlog -sv "$QSYS_SIMDIR/../../top.sv"
```

4. If necessary, add the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the resulting top-level script from the generated simulation directory by specifying the path to `ncsim.sh`.

### Sourcing ModelSim\* Simulator Setup Scripts

Follow these steps to incorporate the generated ModelSim IP simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.tcl`.

```
# # Start of template  
# # If the copied and modified template file is "mentor.do", run it  
# # as: vsim -c -do mentor.do  
# #  
# # Source the generated sim script  
# source msim_setup.tcl  
# # Compile eda/sim_lib contents first  
# dev_com  
# # Override the top-level name (so that elab is useful)  
# set TOP_LEVEL_NAME top  
# # Compile the standalone IP.  
# com  
# # Compile the top-level  
# vlog -sv ../../top.sv  
# # Elaborate the design.  
# elab  
# # Run the simulation  
# run -a  
# # Report success to the shell  
# exit -code 0  
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template  
# If the copied and modified template file is "mentor.do", run it  
# as: vsim -c -do mentor.do  
#  
# Source the generated sim script source msim_setup.tcl  
# Compile eda/sim_lib contents first  
dev_com  
# Override the top-level name (so that elab is useful)  
set TOP_LEVEL_NAME top  
# Compile the standalone IP.  
com  
# Compile the top-level vlog -sv ../../top.sv  
# Elaborate the design.  
elab  
# Run the simulation  
run -a
```



```
# Report success to the shell
exit -code 0
# End of template
```

3. Modify the `TOP_LEVEL_NAME` and compilation step appropriately, depending on the simulation's top-level file. For example:

```
set TOP_LEVEL_NAME sim_top vlog -sv ../../sim_top.sv
```

4. If necessary, add the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the resulting top-level script from the generated simulation directory:

```
vsim -c -do <path to sim_top>.tcl
```

### Sourcing VCS\* Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS simulation scripts into a top-level project simulation script.

1. The generated simulation script contains these template lines. Cut and paste the lines preceding the "helper file" into a new executable file. For example, `synopsys_vcs.f`.

```
## Start of template
## If the copied and modified template file is "vcs_sim.sh", run it
## as: ./vcs_sim.sh
##
## Override the top-level name
## specify a command file containing elaboration options
## (system verilog extension, and compile the top-level).
## Override the sim options, so the simulation
## runs forever (until $finish()).
source vcs_setup.sh \
TOP_LEVEL_NAME=top \
USER_DEFINED_ELAB_OPTIONS="-f ../../../../synopsys_vcs.f" \
USER_DEFINED_SIM_OPTIONS=""
##
## helper file: synopsys_vcs.f
+systemverilogext+.sv
../../top.sv
## End of template
```

2. Delete the first two characters of each line (comment and space) for the `vcs.sh` file, as shown below:

```
# Start of template
# If the copied and modified template file is "vcs_sim.sh", run it
# as: ./vcs_sim.sh
#
# Override the top-level name
# specify a command file containing elaboration options
# (system verilog extension, and compile the top-level).
# Override the sim options, so the simulation
# runs forever (until $finish()).
source vcs_setup.sh \
```



```
TOP_LEVEL_NAME=top \  
USER_DEFINED_ELAB_OPTIONS="'-f ../../../../synopsys_vcs.f' \  
USER_DEFINED_SIM_OPTIONS=""
```

3. Delete the first two characters of each line (comment and space) for the `synopsys_vcs.f` file, as shown below:

```
# helper file: synopsys_vcs.f  
+systemverilogext+.sv  
../../../../top.sv  
# End of template
```

4. Modify the `TOP_LEVEL_NAME` and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=sim_top \  

```

5. If necessary, add the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcs_sim.sh`.

### Sourcing VCS\* MX Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS MX simulation scripts for use in top-level project simulation scripts.

1. The generated simulation script contains these template lines. Cut and paste the lines preceding the "helper file" into a new executable file. For example, `vcsmx.sh`.

```
## Start of template  
## If the copied and modified template file is "vcsmx_sim.sh", run  
## it as: ./vcsmx_sim.sh  
##  
## Do the file copy, dev_com and com steps  
# source vcsmx_setup.sh \  
# SKIP_ELAB=1 \  
  
# SKIP_SIM=1  
#  
## Compile the top level module vlogan +v2k  
+systemverilogext+.sv "$QSYS_SIMDIR/../../../../top.sv"  
  
## Do the elaboration and sim steps  
## Override the top-level name  
## Override the sim options, so the simulation runs  
## forever (until $finish()).  
# source vcsmx_setup.sh \  
# SKIP_FILE_COPY=1 \  
# SKIP_DEV_COM=1 \  
# SKIP_COM=1 \  
# TOP_LEVEL_NAME="'-top top' \  
# USER_DEFINED_SIM_OPTIONS=""  
## End of template
```

2. Delete the first two characters of each line (comment and space), as shown below:

```
# Start of template  
# If the copied and modified template file is "vcsmx_sim.sh", run  
# it as: ./vcsmx_sim.sh  
#
```



```
# Do the file copy, dev_com and com steps
source vcsmx_setup.sh \
SKIP_ELAB=1 \
SKIP_SIM=1

# Compile the top level module
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../top.sv"

# Do the elaboration and sim steps
# Override the top-level name
# Override the sim options, so the simulation runs
# forever (until $finish()).
source vcsmx_setup.sh \
SKIP_FILE_COPY=1 \
SKIP_DEV_COM=1 \
SKIP_COM=1 \
TOP_LEVEL_NAME="'-top top'" \
USER_DEFINED_SIM_OPTIONS=""
# End of template
```

3. Modify the `TOP_LEVEL_NAME` and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME="'-top sim_top'" \
```

4. Make the appropriate changes to the compilation of the your top-level file, for example:

```
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../sim_top.sv"
```

5. If necessary, add the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcsmx_sim.sh`.

## 2.8.6 Synthesizing IP Cores in Other EDA Tools

Optionally, use another supported EDA tool to synthesize a design that includes Intel FPGA IP cores. When you generate the IP core synthesis files for use with third-party EDA synthesis tools, you can create an area and timing estimation netlist. To enable generation, turn on **Create timing and resource estimates for third-party EDA synthesis tools** when customizing your IP variation.

The area and timing estimation netlist describes the IP core connectivity and architecture, but does not include details about the true functionality. This information enables certain third-party synthesis tools to better report area and timing estimates. In addition, synthesis tools can use the timing information to achieve timing-driven optimizations and improve the quality of results.

The Intel Quartus Prime software generates the `<variant name>_syn.v` netlist file in Verilog HDL format, regardless of the output file format you specify. If you use this netlist for synthesis, you must include the IP core wrapper file `<variant name>.v` or `<variant name>.vhd` in your Intel Quartus Prime project.

## 2.8.7 Instantiating IP Cores in HDL

Instantiate an IP core directly in your HDL code by calling the IP core name and declaring the IP core's parameters. This approach is similar to instantiating any other module, component, or subdesign. When instantiating an IP core in VHDL, you must include the associated libraries.

### 2.8.7.1 Example Top-Level Verilog HDL Module

Verilog HDL ALTFP\_MULT in Top-Level Module with One Input Connected to Multiplexer.

```

module MF_top (a, b, sel, datab, clock, result);
    input [31:0] a, b, datab;
    input clock, sel;
    output [31:0] result;
    wire [31:0] wire_dataaa;

    assign wire_dataaa = (sel)? a : b;
    altfp_mult inst1
    (.dataaa(wire_dataaa), .datab(datab), .clock(clock), .result(result));

    defparam
        inst1.pipeline = 11,
        inst1.width_exp = 8,
        inst1.width_man = 23,
        inst1.exception_handling = "no";
endmodule

```

### 2.8.7.2 Example Top-Level VHDL Module

VHDL ALTFP\_MULT in Top-Level Module with One Input Connected to Multiplexer.

```

library ieee;
use ieee.std_logic_1164.all;
library altera_mf;
use altera_mf.altera_mf_components.all;

entity MF_top is
    port (clock, sel : in std_logic;
          a, b, datab : in std_logic_vector(31 downto 0);
          result : out std_logic_vector(31 downto 0));
end entity;

architecture arch_MF_top of MF_top is
    signal wire_dataaa : std_logic_vector(31 downto 0);
begin

    wire_dataaa <= a when (sel = '1') else b;

    inst1 : altfp_mult
        generic map (
            pipeline => 11,
            width_exp => 8,
            width_man => 23,
            exception_handling => "no")
        port map (
            dataaa => wire_dataaa,
            datab => datab,
            clock => clock,
            result => result);
end arch_MF_top;

```





## 2.8.8 Support for the IEEE 1735 Encryption Standard

The Intel Quartus Prime Pro Edition software supports the IEEE1735 v1 encryption standard for IP core decryption. The Intel Quartus Prime Standard Edition software does not support this feature.

When you add the following Verilog or VHDL pragma to your RTL, along with the public key, the Intel Quartus Prime software uses the key to decrypt the IP core. To use this feature, use a simulation or synthesis tool that supports the IEEE1735 standard.

### Verilog/SystemVerilog Encryption Pragma:

```
`pragma protect key_keyowner = "Intel Corporation"  
`pragma protect key_method = "rsa"  
`pragma protect key_keyname = "Altera Key1"  
`pragma protect key_block  
<Encrypted session key>
```

### VHDL Encryption Pragma:

```
`protect key_keyowner = "Intel Corporation"  
`protect key_method = "rsa"  
`protect key_keyname = "Altera Key1"  
`protect key_block  
<Encrypted session key>
```

For all languages, include the key value that is available from your sales representative or FAE.

### Related Links

[myAltera.com](http://myAltera.com)

## 2.9 Integrating Other EDA Tools

Optionally integrate supported EDA design entry, synthesis, simulation, physical synthesis, and formal verification tools into the Intel Quartus Prime design flow. The Intel Quartus Prime software supports netlist files from other EDA design entry and synthesis tools. The Intel Quartus Prime software optionally generates various files for use in other EDA tools.

The Intel Quartus Prime software manages EDA tool files and provides the following integration capabilities:

- Compile all RTL and gate-level simulation model libraries for your device, simulator, and design language automatically (**Tools > Launch Simulation Library Compiler**).
- Include files generated by other EDA design entry or synthesis tools in your project as synthesized design files (**Project > Add/Remove File from Project**).
- Automatically generate optional files for board-level verification (**Assignments > Settings > EDA Tool Settings**).



## 2.10 Managing Team-based Projects

The Intel Quartus Prime software supports multiple designers, design iterations, and platforms. Use the following techniques to preserve and track project changes in a team-based environment. These techniques may also be helpful for individual designers.

### Related Links

- [Using External Revision Control](#) on page 78
- [Migrating Projects Across Operating Systems](#) on page 79

### 2.10.1 Preserving Compilation Results

The Intel Quartus Prime software allows you to transfer your compiled databases from one version of the software to a newer version of the software.

You can export the results of compilation at various stages of the compilation flow, such as synthesis, planned, early place, place, route, and finalize snapshots. A *snapshot* is the compilation output of a compiler stage. Import allows you to restore the preserved compilation database and run subsequent stages in the compiler flow.

Export the compilation snapshot by clicking **Project > Export Design**. The exported files are stored in a file with a .qdb extension. Import the snapshot with **Project > Import Design**.

#### 2.10.1.1 Exporting a Design Partition

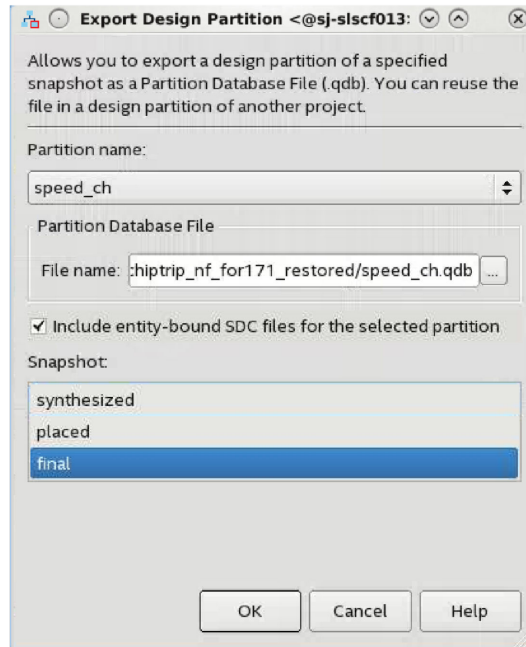
A snapshot preserves the results of each compilation stage. To reuse the snapshot in another project, export the snapshot as a design partition.

Follow these steps to export a snapshot as a design partition:

1. Run Analysis & Synthesis or any stage of the Fitter on your design.
2. Click **Project > Export Design Partition**.
3. Select the **Partition Name** of the entity for export.
4. Select the compilation **Snapshot** for export. The Compiler exports the snapshot as <project>/<partition>.qdb
5. To import the exported partition into another project, open the project and click **Project > Import Design**. Specify the snapshot .qdb file.



Figure 23. Export Design Partition



#### Related Links

[Block-Level Design Flows](#)

### 2.10.2 Factors Affecting Compilation Results

Various changes to project settings, hardware, or software can impact compilation results.

- Project Files—project settings (.qsf, quartus2.ini), design files, and timing constraints (.sdc). Any setting that changes the number of processors during compilation can impact compilation results.
- Hardware—CPU architecture, not including hard disk or memory size differences. Windows XP x32 results are not identical to Windows XP x64 results. Linux x86 results is not identical to Linux x86\_64.
- Intel Quartus Prime Software Version—including build number and installed interim updates. Click **Help > About** to obtain this information.
- Operating System—Windows or Linux operating system, excluding version updates. For example, Windows XP, Windows Vista, and Windows 7 results are identical. Similarly, Linux RHEL, CentOS 4, and CentOS 5 results are identical.

#### Related Links

- [Design Planning for Partial Reconfiguration](#)
- [Power-Up Level](#)

## 2.10.3 Migrating Compilation Results Across Intel Quartus Prime Software Versions

View basic information about your project in the Project Navigator and Compilation Dashboard.

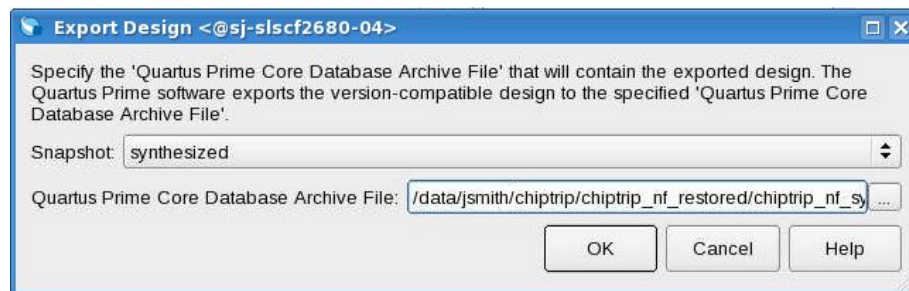
To preserve compilation results for migration to a newer version of the Intel Quartus Prime software, export a version-compatible database file, and then import it into the later version of the Intel Quartus Prime software.

### 2.10.3.1 Exporting the Results Database

Follow these steps to save the compilation results in a version-compatible format for import to a different version of the Intel Quartus Prime software.

1. Open the project for exporting the compilation results in the Intel Quartus Prime software.
2. Generate the project database and netlist with one of the following:
  - Click **Processing** ► **Start** ► **Start Analysis & Synthesis** to generate a post-synthesis netlist.
  - Click **Processing** ► **Start Compilation** to generate a post-fit netlist.
3. Click **Project** ► **Export Design**. Select the **Snapshot** for export. A **Intel Quartus Prime Core Database Archive File** (.qdb) preserves the database. You can select one of the following **Snapshots**:
  - **synthesized**—represents the output of analysis & synthesis.
  - **final**—represents the output of the Fitter.

Figure 24. Export Design Dialog Box



### 2.10.3.2 Importing the Results Database

Follow these steps to import the compilation results from a previous version of the Intel Quartus Prime software to another version of the software.

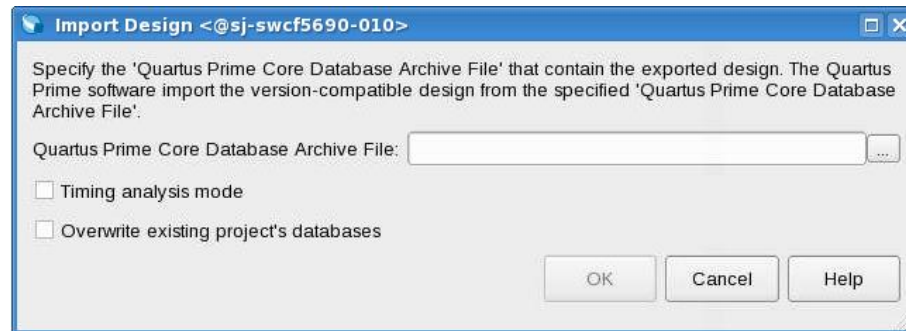
1. In a newer version of the Intel Quartus Prime software, click **New Project Wizard** and create a new project with the same top-level design entity name as the database.
2. Click **Project** ► **Import Design** and specify the **Intel Quartus Prime Core Database Archive File** that contains the exported results.



The **Timing analysis mode** option disables legality checks for certain configuration rules that may have changed from prior versions of the Intel Quartus Prime software. Use this option only if you cannot successfully import your design without it. After you have imported a design in timing analysis mode, you cannot use it to generate programming files.

The **Overwrite existing project's databases** option removes all prior compilation databases from the current project before importing the specified **Core Database Archive File**.

Figure 25. Import Design Dialog Box



## 2.10.4 Archiving Projects

Optionally save the elements of a project in a single, compressed Intel Quartus Prime Archive File (.qar) by clicking **Project > Archive Project**.

The .qar preserves logic design, project, and settings files required to restore the project.

Use this technique to share projects between designers, or to transfer your project to a new version of the Intel Quartus Prime software, or to Intel support. Optionally add compilation results, Platform Designer system files, and third-party EDA tool files to the archive. If you restore the archive in a different version of the Intel Quartus Prime software, you must include the original .qdf in the archive to preserve original compilation results.

### 2.10.4.1 Manually Adding Files To Archives

Follow these steps to add files to an archive manually.

1. Click **Project > Archive Project** and specify the archive file name.
2. Click **Advanced**.
3. Select the **File set** for archive or select **Custom**. Turn on **File subsets** for archive.
4. Click **Add** and select Platform Designer system or EDA tool files. Click **OK**.
5. Click **Archive**.

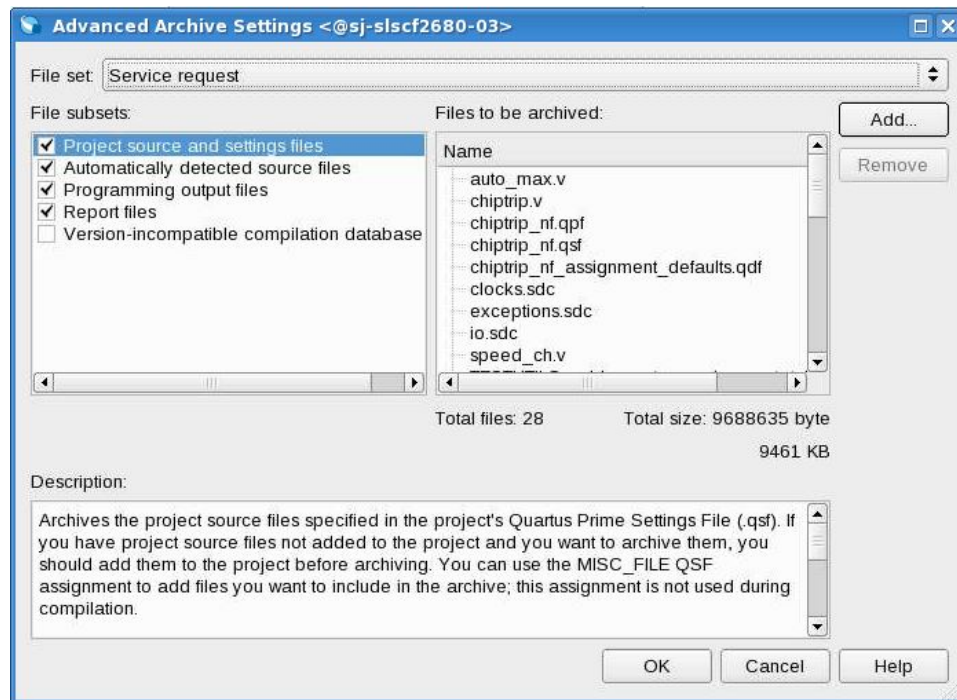
### 2.10.4.2 Archiving Projects for Service Requests

When archiving projects for a service request, include all needed file types for proper debugging by customer support.

To identify and include appropriate archive files for an Intel service request:

1. Click **Project > Archive Project** and specify the archive file name.
2. Click **Advanced**.
3. In **File set**, select **Service request** to include files for Intel Support.
  - Project source and setting files  
(.v, .vhd, .vqm, .qsf, .sdc, .qip, .qpf, .cmp)
  - Automatically detected source files (various)
  - Programming output files (.jdi, .sof, .pof)
  - Report files (.rpt, .pin, .summary, .msg)
4. Click **OK**, and then click **Archive**.

**Figure 26. Archiving Project for Service Request**



### 2.10.5 Using External Revision Control

Your project may involve different team members with distributed responsibilities, such as sub-module design, device and system integration, simulation, and timing closure. In such cases, it may be useful to track and protect file revisions in an external revision control system.

While Intel Quartus Prime project revisions preserve various project setting and constraint combinations, external revision control systems can also track and merge RTL source code, simulation testbenches, and build scripts. External revision control supports design file version experimentation through branching and merging different versions of source code from multiple designers. Refer to your external revision control documentation for setup information.



### 2.10.5.1 Files to Include In External Revision Control

Include the following project file types in external revision control systems:

- Logic design files (.v, .vdh, .bdf, .edf, .vqm)
- Timing constraint files (.sdc)
- Quartus project settings and constraints (.qdf, .qpf, .qsf)
- IP files (.ip, .v, .sv, .vhd, .qip, .sip, .qsys)
- Platform Designer-generated files (.qsys, .ip, .sip)
- EDA tool files (.vo, .vho)

Generate or modify these files manually if you use a scripted design flow. If you use an external source code control system, check-in project files anytime you modify assignments and settings.

### 2.10.6 Migrating Projects Across Operating Systems

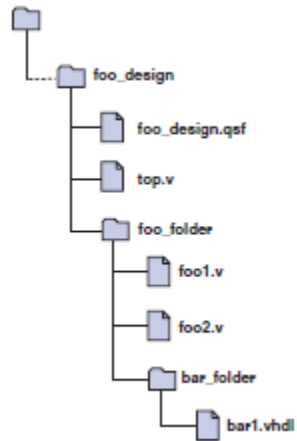
Consider the following cross-platform issues when moving your project from one operating system to another (for example, from Windows to Linux).

#### 2.10.6.1 Migrating Design Files and Libraries

Consider file naming differences when migrating projects across operating systems.

- Use appropriate case for your platform in file path references.
- Use a character set common to both platforms.
- Do not change the forward-slash (/) and back-slash (\) path separators in the .qsf. The Intel Quartus Prime software automatically changes all back-slash (\) path separators to forward-slashes (/) in the .qsf.
- Observe the target platform's file name length limit.
- Use underscore instead of spaces in file and directory names.
- Change library absolute path references to relative paths in the .qsf.
- Ensure that any external project library exists in the new platform's file system.
- Specify file and directory paths as relative to the project directory. For example, for a project titled `foo_design`, specify the source files as: `top.v`, `foo_folder /foo1.v`, `foo_folder /foo2.v`, and `foo_folder /bar_folder/bar1.vhdl`.
- Ensure that all the subdirectories are in the same hierarchical structure and relative path as in the original platform.

Figure 27. All Inclusive Project Directory Structure

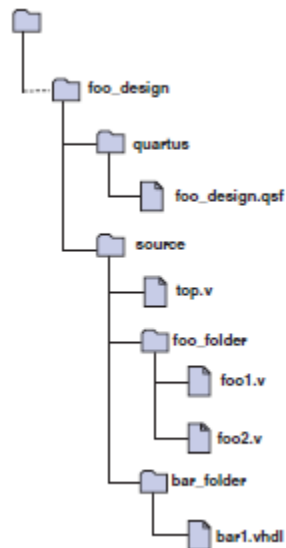


### 2.10.6.1.1 Use Relative Paths

Express file paths using relative path notation (`.. /`).

For example, in the directory structure shown you can specify **top.v** as `../source/top.v` and **foo1.v** as `../source/foo_folder/foo1.v`.

Figure 28. Intel Quartus Prime Project Directory Separate from Design Files



### 2.10.6.2 Design Library Migration Guidelines

The following guidelines apply to library migration across computing platforms:





1. The project directory takes precedence over the project libraries.
2. For Linux, the Intel Quartus Prime software creates the file in the **altera.quartus** directory under the *<home>* directory.
3. All library files are relative to the libraries. For example, if you specify the *user\_lib1* directory as a project library and you want to add the */user\_lib1/fool.v* file to the library, you can specify the *fool.v* file in the *.qsf* as *fool.v*. The Intel Quartus Prime software includes files in specified libraries.
4. If the directory is outside of the project directory, an absolute path is created by default. Change the absolute path to a relative path before migration.
5. When copying projects that include libraries, you must either copy your project library files along with the project directory or ensure that your project library files exist in the target platform.
  - On Windows, the Intel Quartus Prime software searches for the *quartus2.ini* file in the following directories and order:
  - USERPROFILE, for example, *C:\Documents and Settings\<user name>*
  - Directory specified by the TMP environmental variable
  - Directory specified by the TEMP environmental variable
  - Root directory, for example, *C:\*

## 2.11 Scripting API

Optionally use command-line executables or scripts to execute project commands, rather than using the GUI. The following commands are available for scripting project management.

### 2.11.1 Scripting Project Settings

Optionally use a Tcl script to specify settings and constraints, rather than using the GUI. This technique can be helpful if you have many settings and wish to track them in a single file or spreadsheet for iterative comparison. The **.qsf** supports only a limited subset of Tcl commands. Therefore, pass settings and constraints using a Tcl script:

1. Create a text file with the extension **.tcl** that contains your assignments in Tcl format.
2. Source the Tcl script file by adding the following line to the *.qsf*:  

```
set_global_assignment -name SOURCE_TCL_SCR IPT_FILE <file name>.
```

### 2.11.2 Project Revision Commands

Use the following commands for scripting project revisions.

[Create Revision Command](#) on page 82

[Set Current Revision Command](#) on page 82

[Get Project Revisions Command](#) on page 82

[Delete Revision Command](#) on page 82



### 2.11.2.1 Create Revision Command

```
create_revision <name> -based_on <revision_name> -set_current
```

Option	Description
based_on (optional)	Specifies the revision name on which the new revision bases its settings.
set_current (optional)	Sets the new revision as the current revision.

### 2.11.2.2 Set Current Revision Command

The `-force` option enables you to open the revision that you specify under revision name and overwrite the compilation database if the database version is incompatible.

```
set_current_revision -force <revision name>
```

### 2.11.2.3 Get Project Revisions Command

```
get_project_revisions <project_name>
```

### 2.11.2.4 Delete Revision Command

```
delete_revision <revision name>
```

## 2.11.3 Project Archive Commands

Optionally use Tcl commands and the `quartus_sh` executable to create and manage archives of a Quartus project.

### 2.11.3.1 Creating a Project Archive

Use the following command to create a Intel Quartus Prime archive:

```
project_archive <name>.qar
```

You can specify the following other options:

- `-all_revisions` - Includes all revisions of the current project in the archive.
- `-auto_common_directory` - Preserves original project directory structure in archive
- `-common_directory /<name>` - Preserves original project directory structure in specified subdirectory
- `-include_libraries` - Includes libraries in archive
- `-include_outputs` - Includes output files in archive
- `-use_file_set <file_set>` - Includes specified fileset in archive

### 2.11.3.2 Restoring an Archived Project



Use the following Tcl command to restore a Quartus project:

```
project_restore <name>.qar -destination restored -overwrite
```

This example restores to a destination directory named "restored".

## 2.11.4 Project Database Commands

Use the following commands for managing Quartus compilation results:

[Import and Export Version-Compatible Designs from the Design Flow](#) on page 83

[quartus\\_cdb Executables to Manage Version-Compatible Databases](#) on page 83

### 2.11.4.1 Import and Export Version-Compatible Designs from the Design Flow

Optionally use Tcl commands to export and import a full design. You must not open the project or load the database before calling these commands.

These commands require the `quartus_cdb` executable.

- To export a design's snapshot to a file:  
`design::export_design -file <archive.qdb> -snapshot <snapshot_name>`
- To import an exported design's snapshot into a project:  
`design::import_design -file <archive.qdb> [-overwrite] [-timing_analysis_mode]`

The `-overwrite` option removes existing project compilation databases before importing the archived `.qdb` file.

The `-timing_analysis_mode` option is only available for Intel Arria 10 designs. The option disables legality checks for certain configuration rules that may have changed from prior versions of the Intel Quartus Prime software. Use this option only if you cannot successfully import your design without the option. After you import a design in timing analysis mode, you cannot use the imported design to generate programming files.

### 2.11.4.2 quartus\_cdb Executables to Manage Version-Compatible Databases

The command-line arguments to the `quartus_cdb` executable in the Quartus Prime Pro software are `export_design` and `import_design`. The exported version-compatible design files are archived in a file (with a `.qdb` extension). This differs from the Intel Quartus Prime Standard Edition software, which writes all files to a directory.

In the Intel Quartus Prime Standard Edition software, the flow exports both post-map and post-fit databases. In the Intel Quartus Prime Pro Edition software, the export command requires the `snapshot` argument to indicate the target snapshot to export. If the specified snapshot has not been compiled, the flow exits with an error. In ACDS 16.0, export is limited to "synthesized" and "final" snapshots.

```
quartus_cdb <project_name> [-c <revision_name>] --export_design  
--snapshot <snapshot_name> --file <filename>.qdb
```



The import command takes the exported \*.qdb file and the project to which you want to import the design.

```
quartus_cdb <project_name> [-c <revision_name>] --import_design  
--file <archive>.qdb [--overwrite] [--timing_analysis_mode]
```

The --timing\_analysis\_mode option is only available for Intel Arria 10 designs. The option disables legality checks for certain configuration rules that may have changed from prior versions of the Intel Quartus Prime software. Use this option only if you cannot successfully import your design without it. After you have imported a design in timing analysis mode, you cannot use it to generate programming files.

### 2.11.5 Project Library Commands

Use the following commands to script project library changes.

[Specify Project Libraries With SEARCH\\_PATH Assignment](#) on page 84

[Report Specified Project Libraries Commands](#) on page 84

#### 2.11.5.1 Specify Project Libraries With SEARCH\_PATH Assignment

In Tcl, use commands in the ::quartus::project package to specify project libraries, and the set\_global\_assignment command.

Use the following commands to script project library changes:

- set\_global\_assignment -name SEARCH\_PATH "../other\_dir/library1"
- set\_global\_assignment -name SEARCH\_PATH "../other\_dir/library2"
- set\_global\_assignment -name SEARCH\_PATH "../other\_dir/library3"

#### 2.11.5.2 Report Specified Project Libraries Commands

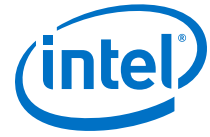
To report any project libraries specified for a project and any global libraries specified for the current installation of the Quartus software, use the get\_global\_assignment and get\_user\_option Tcl commands.

Use the following commands to report specified project libraries:

- get\_global\_assignment -name SEARCH\_PATH
- get\_user\_option -name SEARCH\_PATH

## 2.12 Document Revision History

This document has the following revision history.



**Table 26. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>Revised product branding for Intel standards.</li> <li>Revised topics on Intel FPGA IP Evaluation Mode (formerly OpenCore).</li> <li>Removed <code>-compatible</code> attribute from <code>export_design</code> command content.</li> <li>Updated figure: IP Upgrade Alert in Project Navigator.</li> <li>Updated IP Core Upgrade Status table with new icons, and added row for IP Component Outdated status.</li> </ul>
2017.05.08	17.0.0	<ul style="list-style-type: none"> <li>Added Project Tasks pane and update New Project Wizard.</li> <li>Updated Compilation Dashboard image to show concurrent analysis.</li> <li>Removed Smart Compilation option from Settings dialog box screenshot.</li> <li>Updated IP Catalog screenshots for latest GUIs.</li> <li>Added topic on Back-Annotate Assignments command.</li> <li>Added Exporting a Design Partition topic.</li> <li>Removed mentions to deprecated Incremental Compilation.</li> <li>Added reference to Block-Level Design Flows.</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Added references to compilation stages and snapshots.</li> <li>Removed support for comparing revisions.</li> <li>Added references to <code>.ip</code> file creation during Intel Quartus Prime Pro Edition stand-alone IP generation.</li> <li>Updated IP Core Generation Output files list and diagram.</li> <li>Added Support for IP Core Encryption topic.</li> <li>Rebranding for Intel</li> </ul>
2016.05.03	16.0.0	<ul style="list-style-type: none"> <li>Removed statements about serial equivalence when using multiple processors.</li> <li>Added the "Preserving Compilation Results" section.</li> <li>Added the "Migrating Results Across Quartus Prime Software" section and its subsections for information about importing and exporting compilation results between different versions of Quartus Prime.</li> <li>Added the "Project Database Commands" section and its subsections.</li> </ul>
2016.02.09	15.1.1	<ul style="list-style-type: none"> <li>Clarified instructions for Generating a Combined Simulator Setup Script.</li> <li>Clarified location of <b>Save project output files in specified directory</b> option.</li> </ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>Added Generating Version-Independent IP Simulation Scripts topic.</li> <li>Added example IP simulation script templates for supported simulators.</li> <li>Added Incorporating IP Simulation Scripts in Top-Level Scripts topic.</li> <li>Added Troubleshooting IP Upgrade topic.</li> <li>Updated IP Catalog and parameter editor descriptions for GUI changes.</li> <li>Updated IP upgrade and migration steps for latest GUI changes.</li> <li>Updated Generating IP Cores process for GUI changes.</li> <li>Updated Files Generated for IP Cores and Qsys system description.</li> <li>Removed references to devices and features not supported in version 15.1.</li> <li>Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li> </ul>
<i>continued...</i>		



Date	Version	Changes
2015.05.04	15.0.0	<ul style="list-style-type: none"> <li>Added description of design templates feature.</li> <li>Updated screenshot for DSE II GUI.</li> <li>Added qsys_script IP core instantiation information.</li> <li>Described changes to generating and processing of instance and entity names.</li> <li>Added description of upgrading IP cores at the command line.</li> <li>Updated procedures for upgrading and migrating IP cores.</li> <li>Gate level timing simulation supported only for Cyclone IV and Stratix IV devices.</li> </ul>
2014.12.15	14.1.0	<ul style="list-style-type: none"> <li>Updated content for DSE II GUI and optimizations.</li> <li>Added information about new <b>Assignments &gt; Settings &gt; IP Settings</b> that control frequency of synthesis file regeneration and automatic addition of IP files to the project.</li> </ul>
2014.08.18	14.0a10.0	<ul style="list-style-type: none"> <li>Added information about specifying parameters for IP cores targeting Arria 10 devices.</li> <li>Added information about the latest IP output for version 14.0a10 targeting Arria 10 devices.</li> <li>Added information about individual migration of IP cores to the latest devices.</li> <li>Added information about editing existing IP variations.</li> </ul>
2014.06.30	14.0.0	<ul style="list-style-type: none"> <li>Replaced MegaWizard Plug-In Manager information with IP Catalog.</li> <li>Added standard information about upgrading IP cores.</li> <li>Added standard installation and licensing information.</li> <li>Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor.</li> </ul>
November 2013	13.1.0	<ul style="list-style-type: none"> <li>Conversion to DITA format</li> </ul>
May 2013	13.0.0	<ul style="list-style-type: none"> <li>Overhaul for improved usability and updated information.</li> </ul>
June 2012	12.0.0	<ul style="list-style-type: none"> <li>Removed survey link.</li> <li>Updated information about VERILOG_INCLUDE_FILE.</li> </ul>
November 2011	10.1.1	Template update.
December 2010	10.1.0	<ul style="list-style-type: none"> <li>Changed to new document template.</li> <li>Removed Figure 4-1, Figure 4-6, Table 4-2.</li> <li>Moved "Hiding Messages" to Help.</li> <li>Removed references about the set_user_option command.</li> <li>Removed Classic Timing Analyzer references.</li> </ul>

**Related Links**

[Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 3 Design Planning with the Intel Quartus Prime Software

---

### 3.1 Design Planning with the Intel Quartus Prime Software

Platform planning—the early feasibility analysis of physical constraints—is a fundamental early step in advanced FPGA design. FPGA device densities and complexities are increasing and designs often involve multiple designers. System architects must also resolve design issues when integrating design blocks. However, you can solve potential problems early in the design cycle by following the design planning considerations in this chapter.

*Note:* The Interface Planner helps you to accurately plan constraints for design implementation. Use Interface Planner to prototype interface implementations and rapidly define a legal device floorplan for Intel Arria 10 devices.

Before reading the design planning guidelines discussed in this chapter, consider your design priorities. More device features, density, or performance requirements can increase system cost. Signal integrity and board issues can impact I/O pin locations. Power, timing performance, and area utilization all affect each other. Compilation time is affected when optimizing these priorities.

The Intel Quartus Prime software optimizes designs for the best overall results; however, you can change the settings to better optimize one aspect of your design, such as power utilization. Certain tools or debugging options can lead to restrictions in your design flow. Your design priorities help you choose the tools, features, and methodologies to use for your design.

After you select a device family, to check if additional guidelines are available, refer to the design guidelines section of the appropriate device documentation.

#### Related Links

[Interface Planning](#)

### 3.2 Creating Design Specifications

Before you create your design logic or complete your system design, create detailed design specifications that define the system, specify the I/O interfaces for the FPGA, identify the different clock domains, and include a block diagram of basic design functions.

In addition, creating a test plan helps you to design for verification and ease of manufacture. For example, you might need to validate interfaces incorporated in your design. To perform any built-in self-test functions to drive interfaces, you can use a UART interface with a Nios<sup>®</sup> II processor inside the FPGA device.



If more than one designer works on your design, you must consider a common design directory structure or source control system to make design integration easier. Consider whether you want to standardize on an interface protocol for each design block.

#### Related Links

- [Planning for On-Chip Debugging Tools](#) on page 94
- [Using Platform Designer and Standard Interfaces in System Design](#) on page 88  
For improved reusability and ease of integration.

### 3.3 Selecting Intellectual Property Cores

Intel and its third-party intellectual property (IP) partners offer a large selection of standardized IP cores optimized for Intel devices. The IP you select often affects system design, especially if the FPGA interfaces with other devices in the system. Consider which I/O interfaces or other blocks in your system design are implemented using IP cores, and plan to incorporate these cores in your FPGA design.

The Intel FPGA IP Evaluation Mode, which is available for many IP cores, allows you to program the FPGA to verify your design in the hardware before you purchase the IP license. The evaluation supports the following modes:

- Untethered—the design runs for a limited time.
- Tethered—the design requires an Intel serial JTAG cable connected between the JTAG port on your board and a host computer running the Intel Quartus Prime Programmer for the duration of the hardware evaluation period.

#### Related Links

##### [Intellectual Property](#)

For descriptions of available IP cores.

### 3.4 Using Platform Designer and Standard Interfaces in System Design

You can use the Intel Quartus Prime Platform Designer system integration tool to create your design with fast and easy system-level integration. With Platform Designer, you can specify system components in a GUI and generate the required interconnect logic automatically, along with adapters for clock crossing and width differences.

Because system design tools change the design entry methodology, you must plan to start developing your design within the tool. Ensure all design blocks use appropriate standard interfaces from the beginning of the design cycle so that you do not need to make changes later.

Platform Designer components use Avalon<sup>®</sup> standard interfaces for the physical connection of components, and you can connect any logical device (either on-chip or off-chip) that has an Avalon interface. The Avalon Memory-Mapped interface allows a component to use an address mapped read or write protocol that enables flexible topologies for connecting master components to any slave components. The Avalon Streaming interface enables point-to-point connections between streaming components that send and receive data using a high-speed, unidirectional system interconnect between source and sink ports.





In addition to enabling the use of a system integration tool such as Platform Designer, using standard interfaces ensures compatibility between design blocks from different design teams or vendors. Standard interfaces simplify the interface logic to each design block and enable individual team members to test their individual design blocks against the specification for the interface protocol to ease system integration.

#### Related Links

- [System Design with Platform Designer](#)  
For more information about using Platform Designer to improve your productivity.
- [SOPC Builder User Guide](#)  
For more information about SOPC Builder.

## 3.5 Device Selection

The device you choose affects board specification and layout. Use the following guidelines for selecting a device.

Choose the device family that best suits your design requirements. Families differ in cost, performance, logic and memory density, I/O density, power utilization, and packaging. You must also consider feature requirements, such as I/O standards support, high-speed transceivers, global or regional clock networks, and the number of phase-locked loops (PLLs) available in the device.

Each device family has complete documentation, including a data sheet, which documents device features in detail. You can also see a summary of the resources for each device in the **Device** dialog box in the Intel Quartus Prime software.

Carefully study the device density requirements for your design. Devices with more logic resources and higher I/O counts can implement larger and more complex designs, but at a higher cost. Smaller devices use lower static power. Select a device larger than what your design requires if you want to add more logic later in the design cycle to upgrade or expand your design, and reserve logic and memory for on-chip debugging. Consider requirements for types of dedicated logic blocks, such as memory blocks of different sizes, or digital signal processing (DSP) blocks to implement certain arithmetic functions.

If you have older designs that target an Intel device, you can use their resources as an estimate for your design. Compile existing designs in the Intel Quartus Prime software with the **Auto device selected by the Fitter** option in the **Settings** dialog box. Review the resource utilization to learn which device density fits your design. Consider coding style, device architecture, and the optimization options used in the Intel Quartus Prime software, which can significantly affect the resource utilization and timing performance of your design.

#### Related Links

- [Planning for On-Chip Debugging Tools](#) on page 94  
For information about on-chip debugging.
- [Product Selector](#)  
You can refer to the Altera website to help you choose your device.
- [Selector Guides](#)  
You can review important features of each device family in the refer to the Altera website.



- [Devices and Adapters](#)  
For a list of device selection guides.
- [IP and Megafunctions](#)  
For information on how to obtain resource utilization estimates for certain configurations of Intel's FPGA IP, refer to the user guides for Intel FPGA megafunctions and IP MegaCores on the literature page of the Altera website.

### 3.5.1 Device Migration Planning

Determine whether you want to migrate your design to another device density to allow flexibility when your design nears completion. You may want to target a smaller (and less expensive) device and then move to a larger device if necessary to meet your design requirements. Other designers may prototype their design in a larger device to reduce optimization time and achieve timing closure more quickly, and then migrate to a smaller device after prototyping. If you want the flexibility to migrate your design, you must specify these migration options in the Intel Quartus Prime software at the beginning of your design cycle.

Selecting a migration device impacts pin placement because some pins may serve different functions in different device densities or package sizes. If you make pin assignments in the Intel Quartus Prime software, the Pin Migration View in the Pin Planner highlights pins that change function between your migration devices.

## 3.6 Development Kit Selection

In addition to specifying the device you want to target for compilation, you can also specify a target board or a development kit for your design. When you select a development kit, the Intel Quartus Prime software provides a kit reference design, and creates pin assignments for the kit.

You can select a development kit for your new Intel Quartus Prime project from the **New Project Wizard**, or for an existing project by clicking **Assignments > Device**.

### 3.6.1 Specifying a Development Kit for a New Project

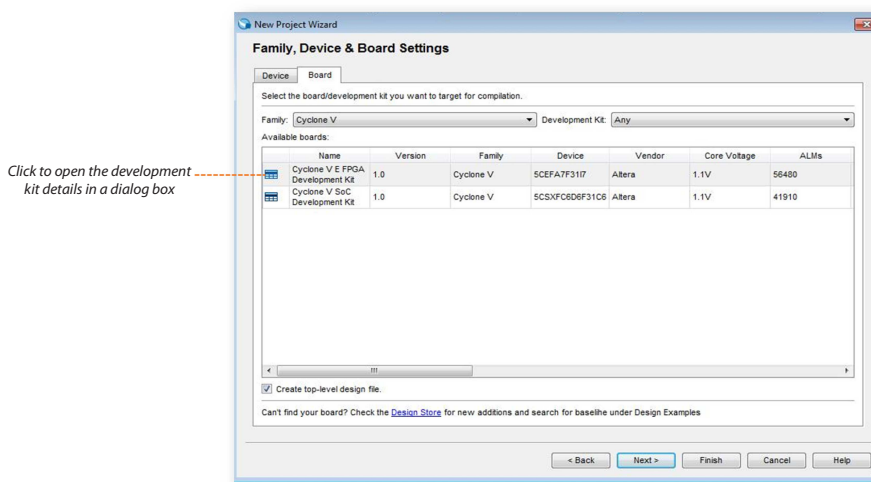
Follow the steps below to select a development kit for a new Intel Quartus Prime project:

1. To open the **New Project Wizard**, click **File > New Project Wizard**.
2. Click the **Board** tab from **Family, Device & Board Settings** page.
3. Select the **Family** and **Development Kit** lists to narrow your board search. The **Available boards** table lists all the available boards for the selected **Family** and **Development Kit** type.
4. To view the development kit details for each of the listed boards, click the icons to the left of the boards in the **Available boards** table. The **Development Kit Details** dialog box appears, displaying all the board details.



5. Select the desired board from the **Available boards** table.
6. To set the selected board design as top-level entity, click the **Create top-level design file** checkbox. This option automatically sets up the pin assignments for the selected board. If you choose to uncheck this option, the Intel Quartus Prime software creates the design for the board and stores the design in `<current_project_dir>/devkits/<design_name>`.
7. Click **Finish**.

**Figure 29. Selecting the Desired Board from New Project Wizard**



**Note:** If you are unable to find the board you are looking for in the **Available Boards** table, click **Design Store** link at the bottom of the page. This link takes you to the design store from where you can purchase development kits and download baseline design examples.

### Related Links

[Design Store](#)

## 3.6.2 Specifying a Development Kit for an Existing Project

Follow the steps below to select a development kit for your existing Intel Quartus Prime project:

1. To open your existing project, click **File > Open Project**.
2. To open the **Device Setting Dialog Box**, click **Assignments > Device**.
3. Select the desired development kit from the **Board** tab and click **OK**.
4. If there are existing pin assignments in your current project, a message box appears, prompting to remove all location assignments. Click **Yes** to remove the **Location** and **I/O Standard** pin assignments. The Intel Quartus Prime software creates the kit's baseline design and stores the design in `<current_project_dir>/devkits/<design_name>`. To retain all your existing pin assignments, click **No**.



**Note:** Repeat the above steps to change the development kit of an existing project.

### 3.6.3 Setting Pin Assignments

The `<design_name>` folder contains the `platform_setup.tcl` file that stores all the pin assignments and the baseline example designs for the board. In addition, the Intel Quartus Prime software creates a `.qdf` file in the `<current_project_dir>` folder, which stores all the default values for the pin assignments.

To manually set up the pin assignments:

1. Click **View ► Tcl Console**.
2. At the Tcl console command prompt, type the command:

```
source <current_project_dir>/devkits/<design_name>/platform_setup.tcl
```

3. At the Tcl console command prompt, type the command:

```
setup_project
```

This command populates all assignments available in the `setup_platform.tcl` file to your `.qsf` file.

## 3.7 Planning for Device Programming or Configuration

System planning includes determining what companion devices, if any, your system requires. Your board layout also depends on the type of programming or configuration method you plan to use for programmable devices. Many programming options require a JTAG interface to connect to the devices, so you might have to set up a JTAG chain on the board. Additionally, the Intel Quartus Prime software uses the settings for the configuration scheme, configuration device, and configuration device voltage to enable the appropriate dual purpose pins as regular I/O pins after you complete configuration. The Intel Quartus Prime software performs voltage compatibility checks of those pins during compilation of your design. Use the **Configuration** tab of the **Device and Pin Options** dialog box to select your configuration scheme.

The device family documentation describes the configuration options available for a device family. For information about programming CPLD devices, refer to your device documentation.

### Related Links

[Configuration Handbook](#)

For more details about configuration options.

## 3.8 Estimating Power

You can use the Intel Quartus Prime power estimation and analysis tools to provide information to PCB board and system designers. Power consumption in FPGA devices depends on the design logic, which can make planning difficult. You can estimate power before you create any source code, or when you have a preliminary version of the design source code, and then perform the most accurate analysis with the Power Analyzer when you complete your design.



You must accurately estimate device power consumption to develop an appropriate power budget and to design the power supplies, voltage regulators, heat sink, and cooling system. Power estimation and analysis helps you satisfy two important planning requirements:

- Thermal—ensure that the cooling solution is sufficient to dissipate the heat generated by the device. The computed junction temperature must fall within normal device specifications.
- Power supply—ensure that the power supplies provide adequate current to support device operation.

The Early Power Estimator (EPE) spreadsheet allows you to estimate power utilization for your design.

You can manually enter data into the EPE spreadsheet, or use the Intel Quartus Prime software to generate device resource information for your design.

To manually enter data into the EPE spreadsheet, enter the device resources, operating frequency, toggle rates, and other parameters for your design. If you do not have an existing design, estimate the number of device resources used in your design, and then enter the data into the EPE spreadsheet manually.

If you have an existing design or a partially completed design, you can use the Intel Quartus Prime software to generate the Early Power Estimator File (.txt, .csv) to assist you in completing the EPE spreadsheet.

The EPE spreadsheet includes the Import Data macro that parses the information in the EPE File and transfers the information into the spreadsheet. If you do not want to use the macro, you can manually transfer the data into the EPE spreadsheet. For example, after importing the EPE File information into the EPE spreadsheet, you can add device resource information. If the existing Intel Quartus Prime project represents only a portion of your full design, manually enter the additional device resources you use in the final design.

Estimating power consumption early in the design cycle allows planning of power budgets and avoids unexpected results when designing the PCB.

When you complete your design, perform a complete power analysis to check the power consumption more accurately. The Power Analyzer tool in the Intel Quartus Prime software provides an accurate estimation of power, ensuring that thermal and supply limitations are met.

#### Related Links

- [Power Analysis](#)  
For more information about power estimation and analysis.
- [Early Power Estimator and Power Analyzer](#)  
The EPE spreadsheets for each supported device family are available on the Altera website.

## 3.9 Selecting Third-Party EDA Tools

Your complete FPGA design flow may include third-party EDA tools in addition to the Intel Quartus Prime software. Determine which tools you want to use with the Intel Quartus Prime software to ensure that they are supported and set up properly, and that you are aware of their capabilities.



### 3.9.1 Synthesis Tool

You can use supported standard third-party EDA synthesis tools to synthesize your Verilog HDL or VHDL design, and then compile the resulting output netlist file in the Intel Quartus Prime software.

Different synthesis tools may give different results for each design. To determine the best tool for your application, you can experiment by synthesizing typical designs for your application and coding style. Perform placement and routing in the Intel Quartus Prime software to get accurate timing analysis and logic utilization results.

The synthesis tool you choose may allow you to create a Intel Quartus Prime project and pass constraints, such as the EDA tool setting, device selection, and timing requirements that you specified in your synthesis project. You can save time when setting up your Intel Quartus Prime project for placement and routing.

Tool vendors frequently add new features, fix tool issues, and enhance performance for Intel devices, you must use the most recent version of third-party synthesis tools.

### 3.9.2 Simulation Tool

Intel provides the Mentor Graphics ModelSim - Intel FPGA Edition with the Intel Quartus Prime software. You can also purchase the ModelSim - Intel FPGA Edition or a full license of the ModelSim software to support large designs and achieve faster simulation performance. The Intel Quartus Prime software can generate both functional and timing netlist files for ModelSim and other third-party simulators.

Use the simulator version that your Intel Quartus Prime software version supports for best results. You must also use the model libraries provided with your Intel Quartus Prime software version. Libraries can change between versions, which might cause a mismatch with your simulation netlist.

### 3.9.3 Formal Verification Tools

Consider whether the Intel Quartus Prime software supports the formal verification tool that you want to use, and whether the flow impacts your design and compilation stages of your design.

Using a formal verification tool can impact performance results because performing formal verification requires turning off certain logic optimizations, such as register retiming, and forces you to preserve hierarchy blocks, which can restrict optimization. Formal verification treats memory blocks as black boxes. Therefore, you must keep memory in a separate hierarchy block so other logic does not get incorporated into the black box for verification. If formal verification is important to your design, plan for limitations and restrictions at the beginning of the design cycle rather than make changes later.

## 3.10 Planning for On-Chip Debugging Tools

Evaluate on-chip debugging tools early in your design process, Making changes to include debugging tools further in the design process is more time consuming and error prone.



In-system debugging tools offer different advantages and trade-offs. A particular debugging tool may work better for different systems and designers. Consider the following debugging requirements when you plan your design:

- JTAG connections—required to perform in-system debugging with JTAG tools. Plan your system and board with JTAG ports that are available for debugging.
- Additional logic resources (ALR)—required to implement JTAG hub logic. If you set up the appropriate tool early in your design cycle, you can include these device resources in your early resource estimations to ensure that you do not overload the device with logic.
- Reserve device memory—required if your tool uses device memory to capture data during system operation. To ensure that you have enough memory resources to take advantage of this debugging technique, consider reserving device memory to use during debugging.
- Reserve I/O pins—required if you use the Logic Analyzer Interface (LAI), which require I/O pins for debugging. If you reserve I/O pins for debugging, you do not have to later change your design or board. The LAI can multiplex signals with design I/O pins if required. Ensure that your board supports a debugging mode, in which debugging signals do not affect system operation.
- Instantiate an IP core in your HDL code—required if your debugging tool uses an Intel FPGA IP core.
- Instantiate the Signal Tap Logic Analyzer IP core—required if you want to manually connect the Signal Tap Logic Analyzer to nodes in your design and ensure that the tapped node names do not change during synthesis.

**Table 27. Factors to Consider When Using Debugging Tools During Design Planning Stages**

Design Planning Factor	Signal Tap Logic Analyzer	System Console	In-System Memory Content Editor	Logic Analyzer Interface (LAI)	Signal Probe	In-System Sources and Probes	Virtual JTAG IP Core
JTAG connections	Yes	Yes	Yes	Yes	—	Yes	Yes
Additional logic resources	—	Yes	—	—	—	—	Yes
Reserve device memory	Yes	Yes	—	—	—	—	—
Reserve I/O pins	—	—	—	Yes	Yes	—	—
Instantiate IP core in your HDL code	—	—	—	—	—	Yes	Yes

**Related Links**

- [System Debugging Tools Overview](#)  
In *Intel Quartus Prime Pro Edition Handbook Volume 3*
- [Design Debugging Using the Signal Tap Logic Analyzer](#)  
In *Intel Quartus Prime Pro Edition Handbook Volume 3*

### 3.11 Design Practices and HDL Coding Styles

When you develop complex FPGA designs, design practices and coding styles have an enormous impact on the timing performance, logic utilization, and system reliability of your device.

### 3.11.1 Design Recommendations

Use synchronous design practices to consistently meet your design goals. Problems with asynchronous design techniques include reliance on propagation delays in a device, incomplete timing analysis, and possible glitches.

In a synchronous design, a clock signal triggers all events. When you meet all register timing requirements, a synchronous design behaves in a predictable and reliable manner for all process, voltage, and temperature (PVT) conditions. You can easily target synchronous designs to different device families or speed grades.

Clock signals have a large effect on the timing accuracy, performance, and reliability of your design. Problems with clock signals can cause functional and timing problems in your design. Use dedicated clock pins and clock routing for best results, and if you have PLLs in your target device, use the PLLs for clock inversion, multiplication, and division. For clock multiplexing and gating, use the dedicated clock control block or PLL clock switchover feature instead of combinational logic, if these features are available in your device. If you must use internally-generated clock signals, register the output of any combinational logic used as a clock signal to reduce glitches.

Consider the architecture of the device you choose so that you can use specific features in your design. For example, the control signals should use the dedicated control signals in the device architecture. Sometimes, you might need to limit the number of different control signals used in your design to achieve the best results.

#### Related Links

- [Recommended Design Practices](#) on page 152
- [www.sunburst-design.com/papers](http://www.sunburst-design.com/papers)  
You can also refer to industry papers for more information about multiple clock design. For a good analysis, refer to *Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs*

### 3.11.2 Recommended HDL Coding Styles

HDL coding styles can have a significant effect on the quality of results for programmable logic designs.

If you design memory and DSP functions, you must understand the target architecture of your device so you can use the dedicated logic block sizes and configurations. Follow the coding guidelines for inferring megafunctions and targeting dedicated device hardware, such as memory and DSP blocks.

#### Related Links

[Recommended HDL Coding Styles](#) on page 100

### 3.11.3 Managing Metastability

Metastability problems can occur in digital design when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the designer cannot guarantee that the signal meets the setup and hold time requirements during the signal transfer.





Designers commonly use a synchronization chain to minimize the occurrence of metastable events. Ensure that your design accounts for synchronization between any asynchronous clock domains. Consider using a synchronizer chain of more than two registers for high-frequency clocks and frequently-toggling data signals to reduce the chance of a metastability failure.

You can use the Intel Quartus Prime software to analyze the average mean time between failures (MTBF) due to metastability when a design synchronizes asynchronous signals, and optimize your design to improve the metastability MTBF. The MTBF due to metastability is an estimate of the average time between instances when metastability could cause a design failure. A high MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design. Determine an acceptable target MTBF given the context of your entire system and the fact that MTBF calculations are statistical estimates.

The Intel Quartus Prime software can help you determine whether you have enough synchronization registers in your design to produce a high enough MTBF at your clock and data frequencies.

#### Related Links

[Managing Metastability with the Intel Quartus Prime Software](#) on page 986

For information about metastability analysis, reporting, and optimization features in the Intel Quartus Prime software

## 3.12 Running Fast Synthesis

You save time when you find design issues early in the design cycle rather than in the final timing closure stages. When the first version of the design source code is complete, you might want to perform a quick compilation to create a kind of silicon virtual prototype (SVP) that you can use to perform timing analysis.

If you synthesize with the Intel Quartus Prime software, you can choose to perform a **Fast** synthesis, which reduces the compilation time, but may give reduced quality of results.

If you design individual design blocks or partitions separately, you can use the Fast synthesis and early timing estimate features as you develop your design. Any issues highlighted in the lower-level design blocks are communicated to the system architect. Resolving these issues might require allocating additional device resources to the individual partition, or changing the timing budget of the partition.

#### Related Links

[Synthesis Effort logic option](#)

For more information about Fast synthesis, refer to Intel Quartus Prime Help.



### 3.13 Document Revision History

Table 28. Document Revision History

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>Changed instances of OpenCore Plus to Intel FPGA IP Evaluation Mode.</li> <li>Changed instances of Qsys to Platform Designer (Standard)</li> </ul>
2017.05.08	17.0.0	<ul style="list-style-type: none"> <li>Removed mentions to Integrated Synthesis.</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Implemented Intel rebranding.</li> </ul>
2016.05.03	16.0.0	Added information about Development Kit selection.
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>Added references to Interface Planning chapter.</li> <li>Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li> </ul>
2015.05.04	15.0.0	Remove support for Early Timing Estimate feature.
2014.06.30	14.0.0	Updated document format.
November 2013	13.1.0	Removed HardCopy device information.
November, 2012	12.1.0	Update for changes to early pin planning feature
June 2012	12.0.0	Editorial update.
November 2011	11.0.1	Template update.
May 2011	11.0.0	<ul style="list-style-type: none"> <li>Added link to System Design with Qsys in "Creating Design Specifications" on page 1-2</li> <li>Updated "Simultaneous Switching Noise Analysis" on page 1-8</li> <li>Updated "Planning for On-Chip Debugging Tools" on page 1-10</li> <li>Removed information from "Planning Design Partitions and Floorplan Location Assignments" on page 1-15</li> </ul>
December 2010	10.1.0	<ul style="list-style-type: none"> <li>Changed to new document template</li> <li>Updated "System Design and Standard Interfaces" on page 1-3 to include information about the Qsys system integration tool</li> <li>Added link to the Product Selector in "Device Selection" on page 1-3</li> <li>Converted information into new table (Table 1-1) in "Planning for On-Chip Debugging Options" on page 1-10</li> <li>Simplified description of incremental compilation usages in "Incremental Compilation with Design Partitions" on page 1-14</li> <li>Added information about the Rapid Recompile option in "Flat Compilation Flow with No Design Partitions" on page 1-14</li> <li>Removed details and linked to Intel Quartus Prime Help in "Fast Synthesis and Early Timing Estimation" on page 1-16</li> </ul>

*continued...*



Date	Version	Changes
July 2010	10.0.0	<ul style="list-style-type: none"> <li>Added new section "System Design" on page 1-3</li> <li>Removed details about debugging tools from "Planning for On-Chip Debugging Options" on page 1-10 and referred to other handbook chapters for more information</li> <li>Updated information on recommended design flows in "Incremental Compilation with Design Partitions" on page 1-14 and removed "Single-Project Versus Multiple-Project Incremental Flows" heading</li> <li>Merged the "Planning Design Partitions" section with the "Creating a Design Floorplan" section. Changed heading title to "Planning Design Partitions and Floorplan Location Assignments" on page 1-15</li> <li>Removed "Creating a Design Floorplan" section</li> <li>Removed "Referenced Documents" section</li> <li>Minor updates throughout chapter</li> </ul>
November 2009	9.1.0	<ul style="list-style-type: none"> <li>Added details to "Creating Design Specifications" on page 1-2</li> <li>Added details to "Intellectual Property Selection" on page 1-2</li> <li>Updated information on "Device Selection" on page 1-3</li> <li>Added reference to "Device Migration Planning" on page 1-4</li> <li>Removed information from "Planning for Device Programming or Configuration" on page 1-4</li> <li>Added details to "Early Power Estimation" on page 1-5</li> <li>Updated information on "Early Pin Planning and I/O Analysis" on page 1-6</li> <li>Updated information on "Creating a Top-Level Design File for I/O Analysis" on page 1-8</li> <li>Added new "Simultaneous Switching Noise Analysis" section</li> <li>Updated information on "Synthesis Tools" on page 1-9</li> <li>Updated information on "Simulation Tools" on page 1-9</li> <li>Updated information on "Planning for On-Chip Debugging Options" on page 1-10</li> <li>Added new "Managing Metastability" section</li> <li>Changed heading title "Top-Down Versus Bottom-Up Incremental Flows" to "Single-Project Versus Multiple-Project Incremental Flows"</li> <li>Updated information on "Creating a Design Floorplan" on page 1-18</li> <li>Removed information from "Fast Synthesis and Early Timing Estimation" on page 1-18</li> </ul>
March 2009	9.0.0	<ul style="list-style-type: none"> <li>No change to content</li> </ul>
November 2008	8.1.0	<ul style="list-style-type: none"> <li>Changed to 8-1/2 x 11 page size. No change to content.</li> </ul>
May 2008	8.0.0	<ul style="list-style-type: none"> <li>Organization changes</li> <li>Added "Creating Design Specifications" section</li> <li>Added reference to new details in the In-System Design Debugging section of volume 3</li> <li>Added more details to the "Design Practices and HDL Coding Styles" section</li> <li>Added references to the new Best Practices for Incremental Compilation and Floorplan Assignments chapter</li> <li>Added reference to the Intel Quartus Prime Language Templates</li> </ul>

### Related Links

#### Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 4 Recommended HDL Coding Styles

---

This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

HDL coding styles have a significant effect on the quality of results for programmable logic designs. Synthesis tools optimize HDL code for both logic utilization and performance; however, synthesis tools cannot interpret the intent of your design. Therefore, the most effective optimizations require conformance to recommended coding styles.

*Note:* For style recommendations, options, or HDL attributes specific to your synthesis tool (including other Quartus software products and other EDA tools), refer to the synthesis tool vendor's documentation.

### Related Links

- [Advanced Synthesis Cookbook](#)
- [Design Examples](#)
- [Reference Designs](#)

### 4.1 Using Provided HDL Templates

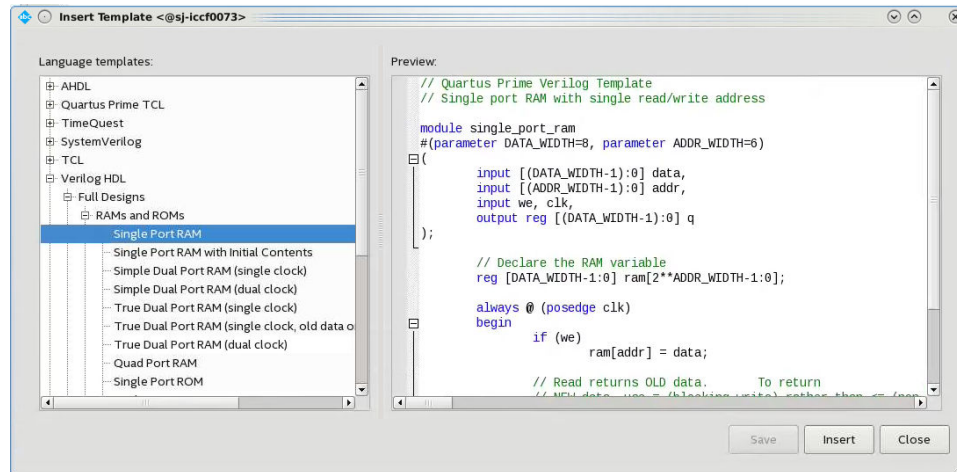
The Intel Quartus Prime software provides templates for Verilog HDL, SystemVerilog, and VHDL templates to start your HDL designs. Many of the HDL examples in this document correspond with the **Full Designs** examples in the **Intel Quartus Prime Templates**. You can insert HDL code into your own design using the templates or examples.

#### 4.1.1 Inserting HDL Code from a Provided Template

1. Click **File** ► **New**.
2. In the **New** dialog box, select the type of design file corresponding to the type of HDL you want to use: **SystemVerilog HDL File**, **VHDL File**, or **Verilog HDL File**; and click **OK**. A text editor tab with a blank file opens.
3. Right-click the blank file, and click **Insert Template...**
4. In the **Insert Template** dialog box, expand the section corresponding to the appropriate HDL, then expand the **Full Designs** section.
5. Select a template. The HDL appears in the **Preview** pane.
6. To paste the HDL design into the blank Verilog or VHDL file you created, click **Insert**.
7. Close the **Insert Template** dialog box by clicking **Close**.



Figure 30. Inserting a RAM Template



**Note:** Use the Intel Quartus Prime Text Editor to modify the HDL design or save the template as an HDL file to edit in your preferred text editor.

## 4.2 Instantiating IP Cores in HDL

Intel provides parameterizable IP cores that are optimized for Intel FPGA device architectures. Using IP cores instead of coding your own logic saves valuable design time.

Additionally, the Intel-provided IP cores offer more efficient logic synthesis and device implementation. Scale the IP core's size and specify various options by setting parameters. To instantiate the IP core directly in your HDL file code, invoke the IP core name and define its parameters as you would do for any other module, component, or subdesign. Alternatively, you can use the IP Catalog (**Tools > IP Catalog**) and parameter editor GUI to simplify customization of your IP core variation. You can infer or instantiate IP cores that optimize device architecture features, for example:

- Transceivers
- LVDS drivers
- Memory and DSP blocks
- Phase-locked loops (PLLs)
- Double-data rate input/output (DDIO) circuitry

For some types of logic functions, such as memories and DSP functions, you can infer device-specific dedicated architecture blocks instead of instantiating an IP core. Intel Quartus Prime synthesis recognizes certain HDL code structures and automatically infers the appropriate IP core or map directly to device atoms.

### Related Links

[Intel FPGA IP Core Literature](#)

## 4.3 Inferring Multipliers and DSP Functions

The following sections describe how to infer multiplier and DSP functions from generic HDL code, and, if applicable, how to target the dedicated DSP block architecture in Intel FPGA devices.

### Related Links

[DSP Solutions Center](#)

### 4.3.1 Inferring Multipliers

To infer multiplier functions, synthesis tools detect multiplier logic and implement this in Intel FPGA IP cores, or map the logic directly to device atoms.

For devices with DSP blocks, Intel Quartus Prime synthesis can implement the function in a DSP block instead of logic, depending on device utilization. The Intel Quartus Prime fitter can also place input and output registers in DSP blocks (that is, perform register packing) to improve performance and area utilization.

The following Verilog HDL and VHDL code examples show that synthesis tools can infer signed and unsigned multipliers as IP cores or DSP block atoms. Each example fits into one DSP block element. In addition, when register packing occurs, no extra logic cells for registers are required.

#### Example 6. Verilog HDL Unsigned Multiplier

```
module unsigned_mult (out, a, b);
    output [15:0] out;
    input [7:0] a;
    input [7:0] b;
    assign out = a * b;
endmodule
```

*Note:* The signed declaration in Verilog HDL is a feature of the Verilog 2001 Standard.

#### Example 7. Verilog HDL Signed Multiplier with Input and Output Registers (Pipelining = 2)

```
module signed_mult (out, clk, a, b);
    output [15:0] out;
    input clk;
    input signed [7:0] a;
    input signed [7:0] b;

    reg signed [7:0] a_reg;
    reg signed [7:0] b_reg;
    reg signed [15:0] out;
    wire signed [15:0] mult_out;

    assign mult_out = a_reg * b_reg;

    always @ (posedge clk)
    begin
        a_reg <= a;
        b_reg <= b;
        out <= mult_out;
    end
endmodule
```



### Example 8. VHDL Unsigned Multiplier with Input and Output Registers (Pipelining = 2)

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY unsigned_mult IS
  PORT (
    a: IN UNSIGNED (7 DOWNTO 0);
    b: IN UNSIGNED (7 DOWNTO 0);
    clk: IN STD_LOGIC;
    aclr: IN STD_LOGIC;
    result: OUT UNSIGNED (15 DOWNTO 0)
  );
END unsigned_mult;

ARCHITECTURE rtl OF unsigned_mult IS
  SIGNAL a_reg, b_reg: UNSIGNED (7 DOWNTO 0);
BEGIN
  PROCESS (clk, aclr)
  BEGIN
    IF (aclr = '1') THEN
      a_reg <= (OTHERS => '0');
      b_reg <= (OTHERS => '0');
      result <= (OTHERS => '0');
    ELSIF (rising_edge(clk)) THEN
      a_reg <= a;
      b_reg <= b;
      result <= a_reg * b_reg;
    END IF;
  END PROCESS;
END rtl;

```

### Example 9. VHDL Signed Multiplier

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY signed_mult IS
  PORT (
    a: IN SIGNED (7 DOWNTO 0);
    b: IN SIGNED (7 DOWNTO 0);
    result: OUT SIGNED (15 DOWNTO 0)
  );
END signed_mult;

ARCHITECTURE rtl OF signed_mult IS
BEGIN
  result <= a * b;
END rtl;

```

## 4.3.2 Inferring Multiply-Accumulator and Multiply-Adder Functions

Synthesis tools detect multiply-accumulator or multiply-adder functions, and either implement them as Intel FPGA IP cores or map them directly to device atoms. During placement and routing, the Intel Quartus Prime software places multiply-accumulator and multiply-adder functions in DSP blocks.

**Note:** Synthesis tools infer multiply-accumulator and multiply-adder functions only if the Intel device family has dedicated DSP blocks that support these functions.

A simple multiply-accumulator consists of a multiplier feeding an addition operator. The addition operator feeds a set of registers that then feeds the second input to the addition operator. A simple multiply-adder consists of two to four multipliers feeding one or two levels of addition, subtraction, or addition/subtraction operators. Addition is always the second-level operator, if it is used. In addition to the multiply-accumulator and multiply-adder, the Intel Quartus Prime Fitter also places input and output registers into the DSP blocks to pack registers and improve performance and area utilization.

Some device families offer additional advanced multiply-adder and accumulator functions, such as complex multiplication, input shift register, or larger multiplications.

The Verilog HDL and VHDL code samples infer multiply-accumulator and multiply-adder functions with input, output, and pipeline registers, as well as an optional asynchronous `clear` signal. Using the three sets of registers provides the best performance through the function, with a latency of three. To reduce latency, remove the registers in your design.

**Note:** To obtain high performance in DSP designs, use register pipelining and avoid unregistered DSP functions.

#### Example 10. Verilog HDL Multiply-Accumulator

```
module sum_of_four_multiply_accumulate
  #(parameter INPUT_WIDTH=18, parameter OUTPUT_WIDTH=44)
  (
    input clk, ena,
    input [INPUT_WIDTH-1:0] dataa, datab, datac, datad,
    input [INPUT_WIDTH-1:0] datae, dataf, datag, datah,
    output reg [OUTPUT_WIDTH-1:0] dataout
  );
  // Each product can be up to 2*INPUT_WIDTH bits wide.
  // The sum of four of these products can be up to 2 bits wider.
  wire [2*INPUT_WIDTH+1:0] mult_sum;

  // Store the results of the operations on the current inputs
  assign mult_sum = (dataa * datab + datac * datad) + \
    (datae * dataf + datag * datah);

  // Store the value of the accumulation
  always @ (posedge clk)
  begin
    if (ena == 1)
    begin
      dataout <= dataout + mult_sum;
    end
  end
endmodule
```

#### Related Links

- [DSP Design Examples](#)
- [AN639: Inferring Stratix V DSP Blocks for FIR Filtering](#)

## 4.4 Inferring Memory Functions from HDL Code

The following coding recommendations provide portable examples of generic HDL code targeting dedicated Intel FPGA memory IP cores. However, if you want to use some of the advanced memory features in Intel FPGA devices, consider using the IP core directly so that you can customize the ports and parameters easily.





You can also use the Intel Quartus Prime templates provided in the Intel Quartus Prime software as a starting point. Most of these designs can also be found on the Design Examples page on the Altera website.

**Table 29. Intel Memory HDL Language Templates**

Language	Full Design Name
VHDL	Single-Port RAM Single-Port RAM with Initial Contents Simple Dual-Port RAM (single clock) Simple Dual-Port RAM (dual clock) True Dual-Port RAM (single clock) True Dual-Port RAM (dual clock) Mixed-Width RAM Mixed-Width True Dual-Port RAM Byte-Enabled Simple Dual-Port RAM Byte-Enabled True Dual-Port RAM Single-Port ROM Dual-Port ROM
Verilog HDL	Single-Port RAM Single-Port RAM with Initial Contents Simple Dual-Port RAM (single clock) Simple Dual-Port RAM (dual clock) True Dual-Port RAM (single clock) True Dual-Port RAM (dual clock) Single-Port ROM Dual-Port ROM
SystemVerilog	Mixed-Width Port RAM Mixed-Width True Dual-Port RAM Mixed-Width True Dual-Port RAM (new data on same port read during write) Byte-Enabled Simple Dual Port RAM Byte-Enabled True Dual-Port RAM

#### Related Links

- [Instantiating IP Cores in HDL](#)  
In *Introduction to Intel FPGA IP Cores*
- [Design Examples](#)
- [Memory](#)  
In *Intel Stratix 10 High-Performance Design Handbook*
- [Embedded Memory Blocks in Intel Arria 10 Devices](#)  
In *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*

### 4.4.1 Inferring RAM functions from HDL Code

To infer RAM functions, synthesis tools recognize certain types of HDL code and map the detected code to technology-specific implementations. For device families that have dedicated RAM blocks, the Intel Quartus Prime software uses an Intel FPGA IP core to target the device memory architecture.

Synthesis tools typically consider all signals and variables that have a multi-dimensional array type and then create a RAM block, if applicable. This is based on the way the signals or variables are assigned or referenced in the HDL source description.



Standard synthesis tools recognize single-port and simple dual-port (one read port and one write port) RAM blocks. Some synthesis tools (such as the Intel Quartus Prime software) also recognize true dual-port (two read ports and two write ports) RAM blocks that map to the memory blocks in certain Intel FPGA devices.

Some tools (such as the Intel Quartus Prime software) also infer memory blocks for array variables and signals that are referenced (read/written) by two indexes, to recognize mixed-width and byte-enabled RAMs for certain coding styles.

**Note:** If your design contains a RAM block that your synthesis tool does not recognize and infer, the design might require a large amount of system memory that can potentially cause compilation problems.

#### 4.4.1.1 Use Synchronous Memory Blocks

Memory blocks in Intel FPGA are synchronous. Therefore, RAM designs must be synchronous to map directly into dedicated memory blocks. For these devices, Intel Quartus Prime synthesis implements asynchronous memory logic in regular logic cells.

Synchronous memory offers several advantages over asynchronous memory, including higher frequencies and thus higher memory bandwidth, increased reliability, and less standby power. To convert asynchronous memory, move registers from the datapath into the memory block.

A memory block is synchronous if it has one of the following read behaviors:

- Memory read occurs in a Verilog HDL `always` block with a `clock` signal or a VHDL clocked process. The recommended coding style for synchronous memories is to create your design with a registered read output.
- Memory read occurs outside a clocked block, but there is a synchronous read address (that is, the address used in the read statement is registered). Synthesis does not always infer this logic as a memory block, or may require external bypass logic, depending on the target device architecture. Avoid this coding style for synchronous memories.

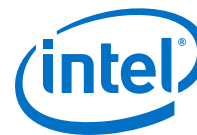
**Note:** The synchronous memory structures in Intel FPGA devices can differ from the structures in other vendors' devices. For best results, match your design to the target device architecture.

This chapter provides coding recommendations for various memory types. All of the examples in this document are synchronous to ensure that they can be directly mapped into the dedicated memory architecture available in Intel FPGAs.

#### 4.4.1.2 Avoid Unsupported Reset and Control Conditions

To ensure correct implementation of HDL code in the target device architecture, avoid unsupported reset conditions or other control logic that does not exist in the device architecture.

The RAM contents of Intel FPGA memory blocks cannot be cleared with a `reset` signal during device operation. If your HDL code describes a RAM with a `reset` signal for the RAM contents, the logic is implemented in regular logic cells instead of a memory block. Do not place RAM read or write operations in an `always` block or `process` block with a `reset` signal. To specify memory contents, initialize the memory or write the data to the RAM during device operation.



In addition to reset signals, other control logic can prevent synthesis from inferring memory logic as a memory block. For example, if you use a clock enable on the read address registers, you can alter the output latch of the RAM, resulting in the synthesized RAM result not matching the HDL description. Use the address stall feature as a read address clock enable to avoid this limitation. Check the documentation for your FPGA device to ensure that your code matches the hardware available in the device.

#### Example 11. Verilog RAM with Reset Signal that Clears RAM Contents: Not Supported in Device Architecture

```

module clear_ram
(
    input clock, reset, we,
    input [7:0] data_in,
    input [4:0] address,
    output reg [7:0] data_out
);

    reg [7:0] mem [0:31];
    integer i;

    always @ (posedge clock or posedge reset)
    begin
        if (reset == 1'b1)
            mem[address] <= 0;
        else if (we == 1'b1)
            mem[address] <= data_in;

        data_out <= mem[address];
    end
endmodule

```

#### Example 12. Verilog RAM with Reset Signal that Affects RAM: Not Supported in Device Architecture

```

module bad_reset
(
    input clock,
    input reset,
    input we,
    input [7:0] data_in,
    input [4:0] address,
    output reg [7:0] data_out,
    input d,
    output reg q
);

    reg [7:0] mem [0:31];
    integer i;

    always @ (posedge clock or posedge reset)
    begin
        if (reset == 1'b1)
            q <= 0;
        else
            begin
                if (we == 1'b1)
                    mem[address] <= data_in;

                data_out <= mem[address];
                q <= d;
            end
    end
endmodule

```

## Related Links

[Specifying Initial Memory Contents at Power-Up](#) on page 120

### 4.4.1.3 Check Read-During-Write Behavior

Ensure the read-during-write behavior of the memory block described in your HDL design is consistent with your target device architecture.

Your HDL source code specifies the memory behavior when you read and write from the same memory address in the same clock cycle. The read returns either the old data at the address, or the new data written to the address. This is referred to as the read-during-write behavior of the memory block. Intel FPGA memory blocks have different read-during-write behavior depending on the target device family, memory mode, and block type.

Synthesis tools preserve the functionality described in your source code. Therefore, if your source code specifies unsupported read-during-write behavior for the RAM blocks, the Intel Quartus Prime software implements the logic in regular logic cells as opposed to the dedicated RAM hardware.

#### Example 13. Continuous read in HDL code

One common problem occurs when there is a continuous read in the HDL code, as in the following examples. Avoid using these coding styles:

```
//Verilog HDL concurrent signal assignment
assign q = ram[raddr_reg];
```

```
-- VHDL concurrent signal assignment
q <= ram(raddr_reg);
```

This type of HDL implies that when a write operation takes place, the read immediately reflects the new data at the address independent of the read clock, which is the behavior of asynchronous memory blocks. Synthesis cannot directly map this behavior to a synchronous memory block. If the write clock and read clock are the same, synthesis can infer memory blocks and add extra bypass logic so that the device behavior matches the HDL behavior. If the write and read clocks are different, synthesis cannot reliably add bypass logic, so it implements the logic in regular logic cells instead of dedicated RAM blocks. The examples in the following sections discuss some of these differences for read-during-write conditions.

In addition, the MLAB memories in certain device logic array blocks (LABs) does not easily support old data or new data behavior for a read-during-write in the dedicated device architecture. Implementing the extra logic to support this behavior significantly reduces timing performance through the memory.

**Note:** For best performance in MLAB memories, ensure that your design does not depend on the read data during a write operation.

In many synthesis tools, you can declare that the read-during-write behavior is not important to your design (for example, if you never read from the same address to which you write in the same clock cycle). In Intel Quartus Prime Pro Edition synthesis, set the synthesis attribute `ramstyle` to `no_rw_check` to allow Intel Quartus Prime software to define the read-during-write behavior of a RAM, rather than use the



behavior specified by your HDL code. This attribute can prevent the synthesis tool from using extra logic to implement the memory block, or can allow memory inference when it would otherwise be impossible.

#### 4.4.1.4 Controlling RAM Inference and Implementation

Intel Quartus Prime synthesis provides options to control RAM inference and implementation for Intel FPGA devices with synchronous memory blocks. Synthesis tools usually do not infer small RAM blocks because implementing small RAM blocks is more efficient if using the registers in regular logic.

To direct the Intel Quartus Prime software to infer RAM blocks globally for all sizes, enable the **Allow Any RAM Size for Recognition** option in the **Advanced Analysis & Synthesis Settings** dialog box.

Alternatively, use the `ramstyle` RTL attribute to specify how an inferred RAM is implemented, including the type of memory block or the use of regular logic instead of a dedicated memory block. Intel Quartus Prime synthesis does not map inferred memory into MLABs unless the HDL code specifies the appropriate `ramstyle` attribute, although the Fitter may map some memories to MLABs.

Set the `ramstyle` attribute in the RTL or in the `.qsf` file.

```
(* ramstyle = "mlab" *) my_shift_reg

set_instance_assignment -name RAMSTYLE_ATTRIBUTE LOGIC -to ram
```

You can also specify the maximum depth of memory blocks for RAM or ROM inference in RTL. Specify the `max_depth` synthesis attribute to the declaration of a variable that represents a RAM or ROM in your design file. For example:

```
// Limit the depth of the memory blocks implement "ram" to 512
// This forces the Intel Quartus Prime software to use two M512 blocks
// instead of one M4K block to implement this RAM
(* max_depth = 512 *) reg [7:0] ram[0:1023];
```

In addition, you can specify the `no_ram` synthesis attribute to prevent RAM inference on a specific array. For example:

```
(* no_ram *) logic [11:0] my_array [0:12];
```

#### Related Links

[Advanced Synthesis Settings](#) on page 230

#### 4.4.1.5 Single-Clock Synchronous RAM with Old Data Read-During-Write Behavior

The code examples in this section show Verilog HDL and VHDL code that infers simple dual-port, single-clock synchronous RAM. Single-port RAM blocks use a similar coding style.

The read-during-write behavior in these examples is to read the old data at the memory address. For best performance in MLAB memories, use the appropriate attribute so that your design does not depend on the read data during a write operation. The simple dual-port RAM code samples map directly into Intel synchronous memory.

Single-port versions of memory blocks (that is, using the same read address and write address signals) allow better RAM utilization than dual-port memory blocks, depending on the device family. Refer to the appropriate device handbook for recommendations on your target device.

#### Example 14. Verilog HDL Single-Clock, Simple Dual-Port Synchronous RAM with Old Data Read-During-Write Behavior

```

module single_clk_ram(
    output reg [7:0] q,
    input [7:0] d,
    input [4:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [31:0];

    always @ (posedge clk) begin
        if (we)
            mem[write_address] <= d;
        q <= mem[read_address]; // q doesn't get d in this clock cycle
    end
endmodule

```

#### Example 15. VHDL Single-Clock, Simple Dual-Port Synchronous RAM with Old Data Read-During-Write Behavior

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

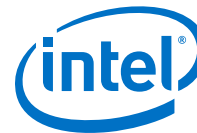
ENTITY single_clock_ram IS
    PORT (
        clock: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END single_clock_ram;

ARCHITECTURE rtl OF single_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ram_block: MEM;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
            q <= ram_block(read_address);
            -- VHDL semantics imply that q doesn't get data
            -- in this clock cycle
        END IF;
    END PROCESS;
END rtl;

```

#### 4.4.1.6 Single-Clock Synchronous RAM with New Data Read-During-Write Behavior

The examples in this section describe RAM blocks in which the read-during-write behavior returns the new value being written at the memory address.



To implement this behavior in the target device, synthesis tools add bypass logic around the RAM block. This bypass logic increases the area utilization of the design, and decreases the performance if the RAM block is part of the design's critical path. If the device memory supports new data read-during-write behavior when in single-port mode (same clock, same read address, and same write address), the Verilog memory block doesn't require any bypass logic. Refer to the appropriate device handbook for specifications on your target device.

The following examples use a blocking assignment for the write so that the data is assigned intermediately.

#### Example 16. Verilog HDL Single-Clock, Simple Dual-Port Synchronous RAM with New Data Read-During-Write Behavior

```
module single_clock_wr_ram(
    output reg [7:0] q,
    input [7:0] d,
    input [6:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [127:0];

    always @ (posedge clk) begin
        if (we)
            mem[write_address] = d;
        q = mem[read_address]; // q does get d in this clock
                               // cycle if we is high
    end
endmodule
```

#### Example 17. VHDL Single-Clock, Simple Dual-Port Synchronous RAM with New Data Read-During-Write Behavior:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY single_clock_ram IS
    PORT (
        clock: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 TO 31;
        read_address: IN INTEGER RANGE 0 TO 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
    );
END single_clock_ram;

ARCHITECTURE rtl OF single_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN
    PROCESS (clock)
        VARIABLE ram_block: MEM;
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) := data;
            END IF;
            q <= ram_block(read_address);
            -- VHDL semantics imply that q doesn't get data
            -- in this clock cycle
        END IF;
    END PROCESS;
END rtl;
```

It is possible to create a single-clock RAM by using an `assign` statement to read the address of `mem` and create the output `q`. By itself, the RTL describes new data read-during-write behavior. However, if the RAM output feeds a register in another hierarchy, a read-during-write results in the old data. Synthesis tools may not infer a RAM block if the tool cannot determine which behavior is described, such as when the memory feeds a hard hierarchical partition boundary. Avoid this type of RTL.

#### Example 18. Avoid Verilog Coding Style with Vague read-during-write Behavior

```
reg [7:0] mem [127:0];
reg [6:0] read_address_reg;

always @ (posedge clk) begin
    if (we)
        mem[write_address] <= d;
        read_address_reg <= read_address;
end
assign q = mem[read_address_reg];
```

#### Example 19. Avoid VHDL Coding Style with Vague read-during-write Behavior

The following example uses a concurrent signal assignment to read from the RAM, and presents a similar behavior.

```
ARCHITECTURE rtl OF single_clock_rw_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL ram_block: MEM;
    SIGNAL read_address_reg: INTEGER RANGE 0 to 31;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
            read_address_reg <= read_address;
        END IF;
    END PROCESS;
    q <= ram_block(read_address_reg);
END rtl;
```

#### 4.4.1.7 Simple Dual-Port, Dual-Clock Synchronous RAM

With dual-clock designs, synthesis tools cannot accurately infer the read-during-write behavior because it depends on the timing of the two clocks within the target device. Therefore, the read-during-write behavior of the synthesized design is undefined and may differ from your original HDL code.

#### Example 20. Verilog HDL Simple Dual-Port, Dual-Clock Synchronous RAM

```
module simple_dual_port_ram_dual_clock
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=6)
(
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] read_addr, write_addr,
    input we, read_clock, write_clock,
    output reg [(DATA_WIDTH-1):0] q
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
```





```

always @ (posedge write_clock)
begin
    // Write
    if (we)
        ram[write_addr] <= data;
    end

always @ (posedge read_clock)
begin
    // Read
    q <= ram[read_addr];
    end

endmodule

```

### Example 21. VHDL Simple Dual-Port, Dual-Clock Synchronous RAM

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dual_clock_ram IS
    PORT (
        clock1, clock2: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
    );
END dual_clock_ram;
ARCHITECTURE rtl OF dual_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL ram_block: MEM;
    SIGNAL read_address_reg : INTEGER RANGE 0 to 31;
BEGIN
    PROCESS (clock1)
    BEGIN
        IF (rising_edge(clock1)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
        END IF;
    END PROCESS;
    PROCESS (clock2)
    BEGIN
        IF (rising_edge(clock2)) THEN
            q <= ram_block(read_address_reg);
            read_address_reg <= read_address;
        END IF;
    END PROCESS;
END rtl;

```

#### Related Links

[Check Read-During-Write Behavior](#) on page 108

#### 4.4.1.8 True Dual-Port Synchronous RAM

The code examples in this section show Verilog HDL and VHDL code that infers true dual-port synchronous RAM. Different synthesis tools may differ in their support for these types of memories.

Intel FPGA synchronous memory blocks have two independent address ports, allowing for operations on two unique addresses simultaneously. A read operation and a write operation can share the same port if they share the same address.

The Intel Quartus Prime software infers true dual-port RAMs in Verilog HDL and VHDL, with the following characteristics:

- Any combination of independent read or write operations in the same clock cycle.
- At most two unique port addresses.
- In one clock cycle, with one or two unique addresses, they can perform:
  - Two reads and one write
  - Two writes and one read
  - Two writes and two reads

In the synchronous RAM block architecture, there is no priority between the two ports. Therefore, if you write to the same location on both ports at the same time, the result is indeterminate in the device architecture. You must ensure your HDL code does not imply priority for writes to the memory block, if you want the design to be implemented in a dedicated hardware memory block. For example, if both ports are defined in the same process block, the code is synthesized and simulated sequentially so that there is a priority between the two ports. If your code does imply a priority, the logic cannot be implemented in the device RAM blocks and is implemented in regular logic cells. You must also consider the read-during-write behavior of the RAM block to ensure that it can be mapped directly to the device RAM architecture.

When a read and write operation occurs on the same port for the same address, the read operation may behave as follows:

- **Read new data**—Intel Arria 10 and Intel Stratix 10 devices support this behavior.
- **Read old data**—Not supported.

When a read and write operation occurs on different ports for the same address (also known as mixed port), the read operation may behave as follows:

- **Read new data**—Intel Quartus Prime Pro Edition synthesis supports this mode by creating bypass logic around the synchronous memory block.
- **Read old data**—Intel Arria 10 and Intel Cyclone 10 devices support this behavior.
- **Read don't care**—Synchronous memory blocks support this behavior in simple dual-port mode.

The Verilog HDL single-clock code sample maps directly into synchronous Intel Arria 10 memory blocks. When a read and write operation occurs on the same port for the same address, the new data being written to the memory is read. When a read and write operation occurs on different ports for the same address, the old data in the memory is read. Simultaneous writes to the same location on both ports results in indeterminate behavior.

If you generate a dual-clock version of this design describing the same behavior, the inferred memory in the target device presents undefined mixed port read-during-write behavior, because it depends on the relationship between the clocks.

### Example 22. Verilog HDL True Dual-Port RAM with Single Clock

```

module true_dual_port_ram_single_clock
#(parameter DATA_WIDTH = 8, ADDR_WIDTH = 6)
(
    input [(DATA_WIDTH-1):0] data_a, data_b,
    input [(ADDR_WIDTH-1):0] addr_a, addr_b,
    input we_a, we_b, clk,
    output reg [(DATA_WIDTH-1):0] q_a, q_b

```



```

);

// Declare the RAM variable
reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

always @ (posedge clk)
begin // Port a
  if (we_a)
  begin
    ram[addr_a] <= data_a;
    q_a <= data_a;
  end
  else
    q_a <= ram[addr_a];
end
always @ (posedge clk)
begin // Port b
  if (we_b)
  begin
    ram[addr_b] <= data_b;
    q_b <= data_b;
  end
  else
    q_b <= ram[addr_b];
end
endmodule

```

### Example 23. VHDL Read Statement Example

```

-- Port A
process(clk)
begin
  if(rising_edge(clk)) then
    if(we_a = '1') then
      ram(addr_a) := data_a;
    end if;
    q_a <= ram(addr_a);
  end if;
end process;

-- Port B
process(clk)
begin
  if(rising_edge(clk)) then
    if(we_b = '1') then
      ram(addr_b) := data_b;
    end if;
    q_b <= ram(addr_b);
  end if;
end process;

```

The VHDL single-clock code sample maps directly into Intel FPGA synchronous memory. When a read and write operation occurs on the same port for the same address, the new data writing to the memory is read. When a read and write operation occurs on different ports for the same address, the behavior results in old data for Intel Arria 10 and Intel Cyclone 10 devices, and is undefined for Intel Stratix 10 devices. Simultaneous write operations to the same location on both ports results in indeterminate behavior.

If you generate a dual-clock version of this design describing the same behavior, the memory in the target device presents undefined mixed port read-during-write behavior because it depends on the relationship between the clocks.

**Example 24. VHDL True Dual-Port RAM with Single Clock**

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

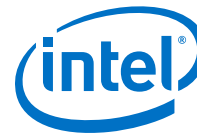
entity true_dual_port_ram_single_clock is
  generic (
    DATA_WIDTH : natural := 8;
    ADDR_WIDTH  : natural := 6
  );
  port (
    clk : in std_logic;
    addr_a : in natural range 0 to 2**ADDR_WIDTH - 1;
    addr_b : in natural range 0 to 2**ADDR_WIDTH - 1;
    data_a : in std_logic_vector((DATA_WIDTH-1) downto 0);
    data_b : in std_logic_vector((DATA_WIDTH-1) downto 0);
    we_a : in std_logic := '1';
    we_b : in std_logic := '1';
    q_a : out std_logic_vector((DATA_WIDTH - 1) downto 0);
    q_b : out std_logic_vector((DATA_WIDTH - 1) downto 0)
  );
end true_dual_port_ram_single_clock;

architecture rtl of true_dual_port_ram_single_clock is
  -- Build a 2-D array type for the RAM
  subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);

  type memory_t is array((2**ADDR_WIDTH - 1) downto 0) of word_t;
  -- Declare the RAM signal.
  signal ram : memory_t;

begin
  process(clk)
  begin
    if(rising_edge(clk)) then -- Port A
      if(we_a = '1') then
        ram(addr_a) <= data_a;
        -- Read-during-write on same port returns NEW data
        q_a <= data_a;
      else
        -- Read-during-write on mixed port returns OLD
        data
          q_a <= ram(addr_a);
        end if;
      end if;
    end process;

    process(clk)
    begin
      if(rising_edge(clk)) then -- Port B
        if(we_b = '1') then
          ram(addr_b) <= data_b;
          -- Read-during-write on same port returns NEW data
          q_b <= data_b;
        else
          -- Read-during-write on mixed port returns OLD data
          q_b <= ram(addr_b);
        end if;
      end if;
    end process;
  end rtl;
```



The port behavior inferred in the Intel Quartus Prime software for the above example is:

```
PORT_A_READ_DURING_WRITE_MODE = "new_data_no_nbe_read"
PORT_B_READ_DURING_WRITE_MODE = "new_data_no_nbe_read"
MIXED_PORT_FEED_THROUGH_MODE = "old"
```

### Related Links

[Guideline: Customize Read-During-Write Behavior](#)

In *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*

#### 4.4.1.9 Mixed-Width Dual-Port RAM

The RAM code examples in this section show SystemVerilog and VHDL code that infers RAM with data ports with different widths.

Verilog-1995 doesn't support mixed-width RAMs because the standard lacks a multi-dimensional array to model the different read width, write width, or both. Verilog-2001 doesn't support mixed-width RAMs because this type of logic requires multiple packed dimensions. Different synthesis tools may differ in their support for these memories. This section describes the inference rules for Intel Quartus Prime Pro Edition synthesis.

The first dimension of the multi-dimensional packed array represents the ratio of the wider port to the narrower port. The second dimension represents the narrower port width. The read and write port widths must specify a read or write ratio supported by the memory blocks in the target device. Otherwise, the synthesis tool does not infer a RAM.

Refer to the Intel Quartus Prime HDL templates for parameterized examples with supported combinations of read and write widths. You can also find examples of true dual port RAMs with two mixed-width read ports and two mixed-width write ports.

#### Example 25. SystemVerilog Mixed-Width RAM with Read Width Smaller than Write Width

```
module mixed_width_ram    // 256x32 write and 1024x8 read
(
    input [7:0] waddr,
    input [31:0] wdata,
    input we, clk,
    input [9:0] raddr,
    output logic [7:0] q
);
    logic [3:0][7:0] ram[0:255];
    always_ff@(posedge clk)
        begin
            if(we) ram[waddr] <= wdata;
            q <= ram[raddr / 4][raddr % 4];
        end
endmodule : mixed_width_ram
```

#### Example 26. SystemVerilog Mixed-Width RAM with Read Width Larger than Write Width

```
module mixed_width_ram    // 1024x8 write and 256x32 read
(
    input [9:0] waddr,
    input [31:0] wdata,
    input we, clk,
    input [7:0] raddr,
    output logic [9:0] q
);
```



```
);
  logic [3:0][7:0] ram[0:255];
  always_ff@(posedge clk)
  begin
    if(we) ram[waddr / 4][waddr % 4] <= wdata;
    q <= ram[raddr];
  end
endmodule : mixed_width_ram
```

### Example 27. VHDL Mixed-Width RAM with Read Width Smaller than Write Width

```
library ieee;
use ieee.std_logic_1164.all;

package ram_types is
  type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
  type ram_t is array (0 to 255) of word_t;
end ram_types;

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.ram_types.all;

entity mixed_width_ram is
  port (
    we, clk : in std_logic;
    waddr : in integer range 0 to 255;
    wdata : in word_t;
    raddr : in integer range 0 to 1023;
    q : out std_logic_vector(7 downto 0));
end mixed_width_ram;

architecture rtl of mixed_width_ram is
  signal ram : ram_t;
begin -- rtl
  process(clk, we)
  begin
    if(rising_edge(clk)) then
      if(we = '1') then
        ram(waddr) <= wdata;
      end if;
      q <= ram(raddr / 4)(raddr mod 4);
    end if;
  end process;
end rtl;
```

### Example 28. VHDL Mixed-Width RAM with Read Width Larger than Write Width

```
library ieee;
use ieee.std_logic_1164.all;

package ram_types is
  type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
  type ram_t is array (0 to 255) of word_t;
end ram_types;

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.ram_types.all;

entity mixed_width_ram is
  port (
    we, clk : in std_logic;
    waddr : in integer range 0 to 1023;
    wdata : in std_logic_vector(7 downto 0);
    raddr : in integer range 0 to 255;
```



```

        q      : out word_t);
end mixed_width_ram;

architecture rtl of mixed_width_ram is
    signal ram : ram_t;
begin -- rtl
    process(clk, we)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                ram(waddr / 4)(waddr mod 4) <= wdata;
            end if;
            q <= ram(raddr);
        end if;
    end process;
end rtl;

```

#### 4.4.1.10 RAM with Byte-Enable Signals

The RAM code examples in this section show SystemVerilog and VHDL code that infers RAM with controls for writing single bytes into the memory word, or byte-enable signals.

Synthesis models byte-enable signals by creating write expressions with two indexes, and writing part of a RAM "word." With these implementations, you can also write more than one byte at once by enabling the appropriate byte enables.

Verilog-1995 doesn't support mixed-width RAMs because the standard lacks a multi-dimensional array to model the different read width, write width, or both. Verilog-2001 doesn't support mixed-width RAMs because this type of logic requires multiple packed dimensions. Different synthesis tools may differ in their support for these memories. This section describes the inference rules for Intel Quartus Prime Pro Edition synthesis.

Refer to the Intel Quartus Prime HDL templates for parameterized examples that you can use for different address widths, and true dual port RAM examples with two read ports and two write ports.

#### Example 29. SystemVerilog Simple Dual-Port Synchronous RAM with Byte Enable

```

module byte_enabled_simple_dual_port_ram
(
    input we, clk,
    input [5:0] waddr, raddr, // address width = 6
    input [3:0] be,          // 4 bytes per word
    input [31:0] wdata,      // byte width = 8, 4 bytes per word
    output reg [31:0] q      // byte width = 8, 4 bytes per word
);
    // use a multi-dimensional packed array
    //to model individual bytes within the word
    logic [3:0][7:0] ram[0:63]; // # words = 1 << address width

    always_ff@(posedge clk)
    begin
        if(we) begin
            if(be[0]) ram[waddr][0] <= wdata[7:0];
            if(be[1]) ram[waddr][1] <= wdata[15:8];
            if(be[2]) ram[waddr][2] <= wdata[23:16];
            if(be[3]) ram[waddr][3] <= wdata[31:24];
        end
        q <= ram[raddr];
    end
endmodule

```

### Example 30. VHDL Simple Dual-Port Synchronous RAM with Byte Enable

```

library ieee;
use ieee.std_logic_1164.all;
library work;

entity byte_enabled_simple_dual_port_ram is
port (
    we, clk : in  std_logic;
    waddr, raddr : in  integer range 0 to 63 ;    -- address width = 6
    be      : in  std_logic_vector(3 downto 0);  -- 4 bytes per word
    wdata   : in  std_logic_vector(31 downto 0); -- byte width = 8
    q       : out std_logic_vector(31 downto 0) ); -- byte width = 8
end byte_enabled_simple_dual_port_ram;

architecture rtl of byte_enabled_simple_dual_port_ram is
    -- build up 2D array to hold the memory
    type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
    type ram_t is array (0 to 63) of word_t;

    signal ram : ram_t;
    signal q_local : word_t;

begin -- Re-organize the read data from the RAM to match the output
    unpack: for i in 0 to 3 generate
        q(8*(i+1) - 1 downto 8*i) <= q_local(i);
    end generate unpack;

    process(clk)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                if(be(0) = '1') then
                    ram(waddr)(0) <= wdata(7 downto 0);
                end if;
                if be(1) = '1' then
                    ram(waddr)(1) <= wdata(15 downto 8);
                end if;
                if be(2) = '1' then
                    ram(waddr)(2) <= wdata(23 downto 16);
                end if;
                if be(3) = '1' then
                    ram(waddr)(3) <= wdata(31 downto 24);
                end if;
            end if;
            q_local <= ram(raddr);
        end if;
    end process;
end rtl;

```

#### 4.4.1.11 Specifying Initial Memory Contents at Power-Up

Your synthesis tool may offer various ways to specify the initial contents of an inferred memory. There are slight power-up and initialization differences between dedicated RAM blocks and the MLAB memory, due to the continuous read of the MLAB.

Intel FPGA dedicated RAM block outputs always power-up to zero, and are set to the initial value on the first read. For example, if address 0 is pre-initialized to FF, the RAM block powers up with the output at 0. A subsequent read after power-up from address 0 outputs the pre-initialized value of FF. Therefore, if a RAM powers up and an enable (read enable or clock enable) is held low, the power-up output of 0 maintains until the first valid read cycle. The synthesis tool implements MLAB using registers that power-up to 0, but initialize to their initial value immediately at power-up or reset. Therefore,





the initial value is seen, regardless of the enable status. The Intel Quartus Prime software maps inferred memory to MLABs when the HDL code specifies an appropriate `ramstyle` attribute.

In Verilog HDL, you can use an initial block to initialize the contents of an inferred memory. Intel Quartus Prime Pro Edition synthesis automatically converts the initial block into a Memory Initialization File (.mif) for the inferred RAM.

### Example 31. Verilog HDL RAM with Initialized Contents

```
module ram_with_init(
    output reg [7:0] q,
    input [7:0] d,
    input [4:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [0:31];
    integer i;

    initial begin
        for (i = 0; i < 32; i = i + 1)
            mem[i] = i[7:0];
        end

    always @ (posedge clk) begin
        if (we)
            mem[write_address] <= d;
        q <= mem[read_address];
    end
endmodule
```

Intel Quartus Prime Pro Edition synthesis and other synthesis tools also support the `$readmemb` and `$readmemh` attributes. These attributes allow RAM initialization and ROM initialization work identically in synthesis and simulation.

### Example 32. Verilog HDL RAM Initialized with the readmemb Command

```
reg [7:0] ram[0:15];
initial
begin
    $readmemb("ram.txt", ram);
end
```

In VHDL, you can initialize the contents of an inferred memory by specifying a default value for the corresponding signal. Intel Quartus Prime Pro Edition synthesis automatically converts the default value into a .mif file for the inferred RAM.

### Example 33. VHDL RAM with Initialized Contents

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY ram_with_init IS
    PORT(
        clock: IN STD_LOGIC;
        data: IN UNSIGNED (7 DOWNTO 0);
        write_address: IN integer RANGE 0 to 31;
        read_address: IN integer RANGE 0 to 31;
        we: IN std_logic;
        q: OUT UNSIGNED (7 DOWNTO 0));
END;
```

```

ARCHITECTURE rtl OF ram_with_init IS
    TYPE MEM IS ARRAY(31 DOWNT0 0) OF unsigned(7 DOWNT0 0);
    FUNCTION initialize_ram
        return MEM is
            variable result : MEM;
    BEGIN
        FOR i IN 31 DOWNT0 0 LOOP
            result(i) := to_unsigned(natural(i), natural'(8));
        END LOOP;
        RETURN result;
    END initialize_ram;

    SIGNAL ram_block : MEM := initialize_ram;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
            q <= ram_block(read_address);
        END IF;
    END PROCESS;
END rtl;

```

#### 4.4.2 Inferring ROM Functions from HDL Code

Synthesis tools infer ROMs when a CASE statement exists in which a value is set to a constant for every choice in the CASE statement.

Because small ROMs typically achieve the best performance when they are implemented using the registers in regular logic, each ROM function must meet a minimum size requirement for inference and placement in memory.

For device architectures with synchronous RAM blocks, to infer a ROM block, synthesis must use registers for either the address or the output. When your design uses output registers, synthesis implements registers from the input registers of the RAM block without affecting the functionality of the ROM. If you register the address, the power-up state of the inferred ROM can be different from the HDL design. In this scenario, Intel Quartus Prime synthesis issues a warning.

The following ROM examples map directly to the Intel FPGA memory architecture.

#### Example 34. Verilog HDL Synchronous ROM

```

module sync_rom (clock, address, data_out);
    input clock;
    input [7:0] address;
    output reg [5:0] data_out;
    reg [5:0] data_out;

    always @ (posedge clock)
    begin
        case (address)
            8'b00000000: data_out = 6'b101111;
            8'b00000001: data_out = 6'b110110;
            ...
            8'b11111110: data_out = 6'b000001;
            8'b11111111: data_out = 6'b101010;
        endcase
    end
endmodule

```



### Example 35. VHDL Synchronous ROM

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sync_rom IS
    PORT (
        clock: IN STD_LOGIC;
        address: IN STD_LOGIC_VECTOR(7 downto 0);
        data_out: OUT STD_LOGIC_VECTOR(5 downto 0)
    );
END sync_rom;

ARCHITECTURE rtl OF sync_rom IS
BEGIN
PROCESS (clock)
    BEGIN
    IF rising_edge (clock) THEN
        CASE address IS
            WHEN "00000000" => data_out <= "101111";
            WHEN "00000001" => data_out <= "110110";
            ...
            WHEN "11111110" => data_out <= "000001";
            WHEN "11111111" => data_out <= "101010";
            WHEN OTHERS      => data_out <= "101111";
        END CASE;
    END IF;
    END PROCESS;
END rtl;

```

### Example 36. Verilog HDL Dual-Port Synchronous ROM Using readmemb

```

module dual_port_rom
#(parameter data_width=8, parameter addr_width=8)
(
    input [(addr_width-1):0] addr_a, addr_b,
    input clk,
    output reg [(data_width-1):0] q_a, q_b
);
    reg [data_width-1:0] rom[2*addr_width-1:0];

    initial // Read the memory contents in the file
        //dual_port_rom_init.txt.
    begin
        $readmemb("dual_port_rom_init.txt", rom);
    end

    always @ (posedge clk)
    begin
        q_a <= rom[addr_a];
        q_b <= rom[addr_b];
    end
endmodule

```

### Example 37. VHDL Dual-Port Synchronous ROM Using Initialization Function

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dual_port_rom is
    generic (
        DATA_WIDTH : natural := 8;
        ADDR_WIDTH  : natural := 8
    );
    port (
        clk          : in std_logic;

```

```

        addr_a : in natural range 0 to 2**ADDR_WIDTH - 1;
        addr_b : in natural range 0 to 2**ADDR_WIDTH - 1;
        q_a    : out std_logic_vector((DATA_WIDTH -1) downto 0);
        q_b    : out std_logic_vector((DATA_WIDTH -1) downto 0)
    );
end entity;

architecture rtl of dual_port_rom is
    -- Build a 2-D array type for the ROM
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
    type memory_t is array(2**ADDR_WIDTH - 1 downto 0) of word_t;

    function init_rom
        return memory_t is
        variable tmp : memory_t := (others => (others => '0'));
    begin
        for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
            -- Initialize each address with the address itself
            tmp(addr_pos) := std_logic_vector(to_unsigned(addr_pos,
DATA_WIDTH));
        end loop;
        return tmp;
    end init_rom;

    -- Declare the ROM signal and specify a default initialization value.
    signal rom : memory_t := init_rom;
begin
    process(clk)
    begin
        if (rising_edge(clk)) then
            q_a <= rom(addr_a);
            q_b <= rom(addr_b);
        end if;
    end process;
end rtl;

```

### 4.4.3 Inferring Shift Registers in HDL Code

To infer shift registers in Intel Arria 10 devices, synthesis tools detect a group of shift registers of the same length, and convert them to an Intel FPGA shift register IP core.

For detection, all shift registers must have the following characteristics:

- Use the same clock and clock enable
- No other secondary signals
- Equally spaced taps that are at least three registers apart



Synthesis recognizes shift registers only for device families with dedicated RAM blocks. Intel Quartus Prime Pro Edition synthesis uses the following guidelines:

- The Intel Quartus Prime software determines whether to infer the Intel FPGA shift register IP core based on the width of the registered bus ( $W$ ), the length between each tap ( $L$ ), or the number of taps ( $N$ ).
- If the **Auto Shift Register Recognition** option is set to **Auto**, Intel Quartus Prime Pro Edition synthesis determines which shift registers are implemented in RAM blocks for logic by using:
  - The **Optimization Technique** setting
  - Logic and RAM utilization information about the design
  - Timing information from **Timing-Driven Synthesis**
- If the registered bus width is one ( $W = 1$ ), Intel Quartus Prime synthesis infers shift register IP if the number of taps times the length between each tap is greater than or equal to 64 ( $N \times L > 64$ ).
- If the registered bus width is greater than one ( $W > 1$ ), and the registered bus width times the number of taps times the length between each tap is greater than or equal to 32 ( $W \times N \times L > 32$ ), the Intel Quartus Prime synthesis infers Intel FPGA shift register IP core.
- If the length between each tap ( $L$ ) is not a power of two, Intel Quartus Prime synthesis needs external logic (LEs or ALMs) to decode the read and write counters, because of different sizes of shift registers. This extra decode logic eliminates the performance and utilization advantages of implementing shift registers in memory.

The registers that Intel Quartus Prime synthesis maps to the Intel FPGA shift register IP core, and places in RAM are not available in a Verilog HDL or VHDL output file for simulation tools, because their node names do not exist after synthesis.

*Note:* The Compiler cannot implement a shift register that uses a `shift enable` signal into MLAB memory; instead, the Compiler uses dedicated RAM blocks. To control the type of memory structure that implements the shift register, use the `ramstyle` attribute.

#### 4.4.3.1 Simple Shift Register

The examples in this section show a simple, single-bit wide, 67-bit long shift register.

Intel Quartus Prime synthesis implements the register ( $W = 1$  and  $M = 67$ ) in an `ALTSHIFT_TAPS` IP core for supported devices and maps it to RAM in supported devices, which may be placed in dedicated RAM blocks or MLAB memory. If the length of the register is less than 67 bits, Intel Quartus Prime synthesis implements the shift register in logic.

#### Example 38. Verilog HDL Single-Bit Wide, 64-Bit Long Shift Register

```
module shift_1x67 (clk, shift, sr_in, sr_out);
    input clk, shift;
    input sr_in;
    output sr_out;

    reg [66:0] sr;

    always @ (posedge clk)
    begin
        if (shift == 1'b1)
```

```

begin
    sr[66:1] <= sr[65:0];
    sr[0] <= sr_in;
end
end
assign sr_out = sr[65];
endmodule

```

### Example 39. VHDL Single-Bit Wide, 64-Bit Long Shift Register

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
ENTITY shift_1x67 IS
    PORT (
        clk: IN STD_LOGIC;
        shift: IN STD_LOGIC;
        sr_in: IN STD_LOGIC;
        sr_out: OUT STD_LOGIC
    );
END shift_1x67;

ARCHITECTURE arch OF shift_1x67 IS
    TYPE sr_length IS ARRAY (66 DOWNT0 0) OF STD_LOGIC;
    SIGNAL sr: sr_length;
BEGIN
    PROCESS (clk)
        BEGIN
            IF (rising_edge(clk)) THEN
                IF (shift = '1') THEN
                    sr(66 DOWNT0 1) <= sr(65 DOWNT0 0);
                    sr(0) <= sr_in;
                END IF;
            END IF;
        END PROCESS;
        sr_out <= sr(65);
    END arch;

```

#### 4.4.3.2 Shift Register with Evenly Spaced Taps

The following examples show a Verilog HDL and VHDL 8-bit wide, 64-bit long shift register ( $W > 1$  and  $M = 64$ ) with evenly spaced taps at 15, 31, and 47.

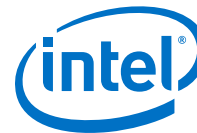
The synthesis software implements this function in a single ALTSHIFT\_TAPS IP core and maps it to RAM in supported devices, which is allowed placement in dedicated RAM blocks or MLAB memory.

### Example 40. Verilog HDL 8-Bit Wide, 64-Bit Long Shift Register with Evenly Spaced Taps

```

module top (clk, shift, sr_in, sr_out, sr_tap_one, sr_tap_two,
    sr_tap_three );
    input clk, shift;
    input [7:0] sr_in;
    output [7:0] sr_tap_one, sr_tap_two, sr_tap_three, sr_out;
    reg [7:0] sr [64:0];
    integer n;
    always @ (posedge clk)
        begin
            if (shift == 1'b1)
                begin
                    for (n = 64; n>0; n = n-1)
                        begin
                            sr[n] <= sr[n-1];
                        end
                    sr[0] <= sr_in;
                end
        end
end

```



```
assign sr_tap_one = sr[16];
assign sr_tap_two = sr[32];
assign sr_tap_three = sr[48];
assign sr_out = sr[64];
endmodule
```

## 4.5 Register and Latch Coding Guidelines

This section provides device-specific coding recommendations for Intel registers and latches. Understanding the architecture of the target Intel device helps ensure that your RTL produces the expected results and achieves the optimal quality of results.

### 4.5.1 Register Power-Up Values

Registers in the device core always power-up to a low (0) logic level on all Intel FPGA devices. However, if your design specifies a power-up level other than 0, synthesis tools can implement logic that causes registers to behave as if they were powering up to a high (1) logic level.

If your design uses a `preset` signal, but your device does not support presets in the register architecture, synthesis may convert the `preset` signal to a `clear` signal, which requires to perform a NOT gate push-back optimization. NOT gate push-back adds an inverter to the input and the output of the register, so that the reset and power-up conditions appear high, and the device operates as expected. In this case, your synthesis tool may issue a message about the power-up condition. The register itself powers up low, but since the register output inverts, the signal that arrives at all destinations is high.

Due to these effects, if you specify a non-zero reset value, your synthesis tool may use the asynchronous clear (`aclr`) signals available on the registers to implement the high bits with NOT gate push-back. In that case, the registers look as though they power-up to the specified reset value.

When an asynchronous load (`aload`) signal is available in the device registers, your synthesis tools can implement a reset of 1 or 0 value by using an asynchronous load of 1 or 0. When the synthesis tool uses a `load` signal, it is not performing NOT gate push-back, so the registers power-up to a 0 logic level.

For additional details, refer to the appropriate device family handbook.

Optionally use an explicit reset signal for the design, which forces all registers into their appropriate values after reset. Use this practice to reset the device after power-up to restore the proper state.

Make your design more stable and avoid potential glitches by synchronizing external or combinational logic of the device architecture before you drive the asynchronous control ports of registers.

#### Related Links

[Recommended Design Practices](#) on page 152

### 4.5.1.1 Specifying a Power-Up Value

To specify a particular power-up condition for your design, use the synthesis options available in your synthesis tool. Intel Quartus Prime Pro Edition synthesis provides the **Power-Up Level** logic option.

You can also specify the power-up level with an `altera_attribute` assignment in your source code. This attribute forces synthesis to perform NOT gate push-back, because synthesis tools cannot actually change the power-up states of core registers.

You can apply the **Power-Up Level** logic option to a specific register, or to a design entity, module, or subdesign. When you assign this option, every register in that block receives the value. Registers power up to 0 by default. Therefore, you can use this assignment to force all registers to power-up to 1 using NOT gate push-back.

Setting the **Power-Up Level** to a logic level of **high** for a large design entity could degrade the quality of results due to the number of inverters that requires. In some situations, this design style causes issues due to `enable` signal inference or secondary control logic inference. It may also be more difficult to migrate this type of designs.

Some synthesis tools can also read the default or initial values for registered signals and implement this behavior in the device. For example, Intel Quartus Prime Pro Edition synthesis converts default values for registered signals into **Power-Up Level** settings. When the Intel Quartus Prime software reads the default values, the synthesized behavior matches the power-up state of the HDL code during a functional simulation.

#### Example 41. Verilog Register with High Power-Up Value

```
reg q = 1'b1; //q has a default value of '1'

always @ (posedge clk)
begin
    q <= d;
end
```

#### Example 42. VHDL Register with High Power-Up Level

```
SIGNAL q : STD_LOGIC := '1'; -- q has a default value of '1'

PROCESS (clk, reset)
BEGIN
    IF (rising_edge(clk)) THEN
        q <= d;
    END IF;
END PROCESS;
```

Your design may contain undeclared default power-up conditions based on signal type. If you declare a VHDL register signal as an integer, Intel Quartus Prime synthesis uses the left end of the integer range as the power-up value. For the default signed integer type, the default power-up value is the highest magnitude negative integer (100...001). For an unsigned integer type, the default power-up value is 0.





**Note:** If the target device architecture does not support two asynchronous control signals, such as `aclr` and `aload`, you cannot set a different power-up state and reset state. If the NOT gate push-back algorithm creates logic to set a register to 1, that register powers-up high. If you set a different power-up condition through a synthesis attribute or initial value, synthesis ignores the power-up level.

#### 4.5.2 Secondary Register Control Signals Such as Clear and Clock Enable

The registers in Intel FPGAs provide a number of secondary control signals. Use these signals to implement control logic for each register without using extra logic cells. Intel FPGA device families vary in their support for secondary signals, so consult the device family data sheet to verify which signals are available in your target device.

To make the most efficient use of the signals in the device, ensure that HDL code matches the device architecture as closely as possible. The control signals have a certain priority due to the nature of the architecture. Your HDL code must follow that priority where possible.

Your synthesis tool can emulate any control signals using regular logic, so achieving functionally correct results is always possible. However, if your design requirements allow flexibility in controlling use and priority of control signals, match your design to the target device architecture to achieve the most efficient results. If the priority of the signals in your design is not the same as that of the target architecture, you may require extra logic to implement the control signals. This extra logic uses additional device resources, and can cause additional delays for the control signals.

In certain cases, using logic other than the dedicated control logic in the device architecture can have a larger impact. For example, the `clock enable` signal has priority over the synchronous `reset` or `clear` signal in the device architecture. The `clock enable` turns off the clock line in the LAB, and the `clear` signal is synchronous. Therefore, in the device architecture, the synchronous clear takes effect only when a clock edge occurs.

If you define a register with a synchronous `clear` signal that has priority over the `clock enable` signal, Intel Quartus Prime synthesis emulates the clock enable functionality using data inputs to the registers. You cannot apply a Clock Enable Multicycle constraint, because the emulated functionality does not use the `clock enable` port of the register. In this case, using a different priority causes unexpected results with an assignment to the `clock enable` signal.

The signal order is the same for all Intel FPGA device families. However, not all device families provide every signal. The priority order is:

1. Asynchronous Clear (`clrn`)—highest priority
2. Enable (`ena`)
3. Synchronous Clear (`sclr`)
4. Synchronous Load (`sload`)
5. Data In (`data`)—lowest priority

The priority order for secondary control signals in Intel FPGA devices differs from the order for other vendors' FPGA devices. If your design requirements are flexible regarding priority, verify that the secondary control signals meet design performance requirements when migrating designs between FPGA vendors. To achieve the best results, try to match your target device architecture.

#### Example 43. Verilog D-type Flipflop bus with Secondary Signals

This module uses all Intel Arria 10 DFF secondary signals: `clrn`, `ena`, `sclr`, and `sload`. Note that it instantiates 8-bit bus of DFFs rather than a single DFF, because synthesis infers some secondary signals only if there are multiple DFFs with the same secondary signal.

```
module top(clk, clrn, sclr, sload, ena, data, sdata, q);
    input clk, clrn, sclr, sload, ena;
    input [7:0] data, sdata;
    output [7:0] q;
    reg [7:0] q;
    always @ (posedge clk or posedge clrn)
        begin
            if (clrn)
                q <= 8'b0;
            else if (ena)
                begin
                    if (sclr)
                        q <= 8'b0;
                    else if (!sload)
                        q <= data;
                    else
                        q <= sdata;
                end
            end
        end
endmodule
```

#### Related Links

##### [Clock Enable Multicycle](#)

In *Intel Quartus Prime Timing Analyzer Cookbook*

### 4.5.3 Latches

A latch is a small combinational loop that holds the value of a signal until a new value is assigned. Synthesis tools can infer latches from HDL code when you did not intend to use a latch. If you do intend to infer a latch, it is important to infer it correctly to guarantee correct device operation.

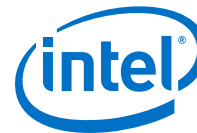
*Note:* Design without the use of latches whenever possible.

#### Related Links

[Avoid Unintended Latch Inference](#) on page 155

#### 4.5.3.1 Avoid Unintentional Latch Generation

When you design combinational logic, certain coding styles can create an unintentional latch. For example, when `CASE` or `IF` statements do not cover all possible input conditions, synthesis tools can infer latches to hold the output if a new output value is not assigned. Check your synthesis tool messages for references to inferred latches.



If your code unintentionally creates a latch, modify your RTL to remove the latch:

- Synthesis infers a latch when HDL code assigns a value to a signal outside of a clock edge (for example, with an asynchronous `reset`), but the code does not assign a value in an edge-triggered design block.
- Unintentional latches also occur when HDL code assigns a value to a signal in an edge-triggered design block, but synthesis optimizations remove that logic. For example, when a `CASE` or `IF` statement tests a condition that only evaluates to `FALSE`, synthesis removes any logic or signal assignment in that statement during optimization. This optimization may result in the inference of a latch for the signal.
- Omitting the final `ELSE` or `WHEN OTHERS` clause in an `IF` or `CASE` statement can also generate a latch. Don't care (`X`) assignments on the default conditions are useful in preventing latch generation. For the best logic optimization, assign the default `CASE` or final `ELSE` value to don't care (`X`) instead of a logic value.

In Verilog HDL designs, use the `full_case` attribute to treat unspecified cases as don't care values (`X`). However, since the `full_case` attribute is synthesis-only, it can cause simulation mismatches, because simulation tools still treat the unspecified cases as latches.

#### Example 44. VHDL Code Preventing Unintentional Latch Creation

Without the final `ELSE` clause, the following code creates unintentional latches to cover the remaining combinations of the `SEL` inputs. When you are targeting a Stratix series device with this code, omitting the final `ELSE` condition can cause synthesis tools to use up to six LEs, instead of the three it uses with the `ELSE` statement. Additionally, assigning the final `ELSE` clause to `1` instead of `X` can result in slightly more LEs, because synthesis tools cannot perform as much optimization when you specify a constant value as opposed to a don't care value.

```
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY nolatch IS
    PORT (a,b,c: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
          oput: OUT STD_LOGIC);
END nolatch;

ARCHITECTURE rtl OF nolatch IS
BEGIN
    PROCESS (a,b,c,sel) BEGIN
        IF sel = "00000" THEN
            oput <= a;
        ELSIF sel = "00001" THEN
            oput <= b;
        ELSIF sel = "00010" THEN
            oput <= c;
        ELSE
            oput <= 'X'; --/ --- Prevents latch inference
        END IF;
    END PROCESS;
END rtl;
```

#### 4.5.3.2 Inferring Latches Correctly

Synthesis tools can infer a latch that does not exhibit the glitch and timing hazard problems typically associated with combinational loops. Intel Quartus Prime Pro Edition software reports latches that synthesis inferred in the **User-Specified and**

**Inferred Latches** section of the Compilation Report. This report indicates whether or not the latch presents a timing hazard, and the total number of user-specified and inferred latches.

*Note:* Timing analysis does not completely model latch timing in some cases. Do not use latches unless required by your design, and you fully understand the impact of using the latches.

If a latch or combinational loop in your design doesn't appear in the **User Specified and Inferred Latches** section, it means that Intel Quartus Prime synthesis didn't infer the latch as a safe latch, so it is not considered glitch-free.

All combinational loops listed in the **Analysis & Synthesis Logic Cells Representing Combinational Loops** table in the Compilation Report are at risk of timing hazards. These entries indicate possible problems with your design that you should investigate. However, it is possible to have a correct design that includes combinational loops. For example, it is possible that the combinational loop cannot be sensitized. This occurs when there is an electrical path in the hardware, but either:

- The designer knows that the circuit never encounters data that causes that path to be activated, or
- The surrounding logic is set up in a mutually exclusive manner that prevents that path from ever being sensitized, independent of the data input.

For 6-input LUT-based devices, Intel Quartus Prime synthesis implements all latch inputs with a single adaptive look-up table (ALUT) in the combinational loop. Therefore, all latches in the **User-Specified and Inferred Latches** table are free of timing hazards when a single input changes.

If Intel Quartus Prime synthesis report lists a latch as a safe latch, other optimizations, such as physical synthesis netlist optimizations in the Fitter, maintain the hazard-free performance. To ensure hazard-free behavior, only one control input can change at a time. Changing two inputs simultaneously, such as deasserting `set` and `reset` at the same time, or changing data and enable at the same time, can produce incorrect behavior in any latch.

Intel Quartus Prime synthesis infers latches from `always` blocks in Verilog HDL and `process` statements in VHDL. However, Intel Quartus Prime synthesis does not infer latches from continuous assignments in Verilog HDL, or concurrent signal assignments in VHDL. These rules are the same as for register inference. The Intel Quartus Prime synthesis infers registers or flipflops only from `always` blocks and `process` statements.

#### Example 45. Verilog HDL Set-Reset Latch

```
module simple_latch (  
    input SetTerm,  
    input ResetTerm,  
    output reg LatchOut  
);  
always @ (SetTerm or ResetTerm) begin  
    if (SetTerm)  
        LatchOut = 1'b1;  
    else if (ResetTerm)  
        LatchOut = 1'b0;  
end  
endmodule
```



### Example 46. VHDL Data Type Latch

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY simple_latch IS
    PORT (
        enable, data    : IN STD_LOGIC;
        q                : OUT STD_LOGIC
    );
END simple_latch;
ARCHITECTURE rtl OF simple_latch IS
BEGIN
    latch : PROCESS (enable, data)
    BEGIN
        IF (enable = '1') THEN
            q <= data;
        END IF;
    END PROCESS latch;
END rtl;

```

The following example shows a Verilog HDL continuous assignment that does not infer a latch in the Intel Quartus Prime software:

### Example 47. Verilog Continuous Assignment Does Not Infer Latch

```

assign latch_out = (~en & latch_out) | (en & data);

```

The behavior of the assignment is similar to a latch, but it may not function correctly as a latch, and its timing is not analyzed as a latch. Intel Quartus Prime Pro Edition synthesis also creates safe latches when possible for instantiations of an Altera latch IP core. Use an Altera latch IP core to define a latch with any combination of data, enable, set, and reset inputs. The same limitations apply for creating safe latches as for inferring latches from HDL code.

Inferring the Altera latch IP core in another synthesis tool ensures that Intel Quartus Prime synthesis also recognizes the implementation as a latch. If a third-party synthesis tool implements a latch using the Altera latch IP core, Intel Quartus Prime Pro Edition synthesis reports the latch in the **User-Specified and Inferred Latches** table, in the same manner as it lists latches you define in HDL source code. The coding style necessary to produce an Altera latch IP core implementation may depend on your synthesis tool. Some third-party synthesis tools list the number of Altera latch IP cores that are inferred.

The Fitter uses global routing for control signals, including signals that synthesis identifies as latch enables. In some cases the global insertion delay may decrease the timing performance. If necessary, you can turn off the **Intel Quartus Prime Global Signal** logic option to manually prevent the use of global signals. The **Global & Other Fast Signals** table in the Compilation Report reports Global latch enables.

## 4.6 General Coding Guidelines

This section describes how coding styles impact synthesis of HDL code into the target Intel FPGA devices. You can improve your design efficiency and performance by following these recommended coding styles, and designing logic structures to match the appropriate device architecture.

### 4.6.1 Tri-State Signals

Use tri-state signals only when they are attached to top-level bidirectional or output pins.

Avoid lower-level bidirectional pins. Also avoid using the z logic value unless it is driving an output or bidirectional pin. Even though some synthesis tools implement designs with internal tri-state signals correctly in Intel FPGA devices using multiplexer logic, do not use this coding style for Intel FPGA designs.

*Note:* In hierarchical block-based design flows, a hierarchical boundary cannot contain any bidirectional ports, unless the lower-level bidirectional port is connected directly through the hierarchy to a top-level output pin without connecting to any other design logic. If you use boundary tri-states in a lower-level block, synthesis software must push the tri-states through the hierarchy to the top level to make use of the tri-state drivers on output pins of Intel FPGA devices. Because pushing tri-states requires optimizing through hierarchies, lower-level tri-states are restricted with block-based design methodologies.

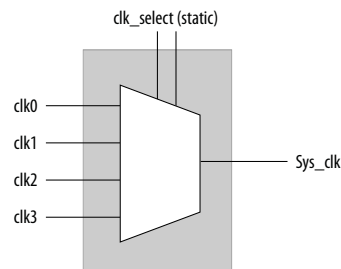
### 4.6.2 Clock Multiplexing

Clock multiplexing is sometimes used to operate the same logic function with different clock sources. This type of logic can introduce glitches that create functional problems. The delay inherent in the combinational logic can also lead to timing problems. Clock multiplexers trigger warnings from a wide range of design rule check and timing analysis tools.

Use dedicated hardware to perform clock multiplexing when it is available, instead of using multiplexing logic. For example, you can use the Clock Switchover feature or the Clock Control Block available in certain Intel FPGA devices. These dedicated hardware blocks avoid glitches, ensure that you use global low-skew routing lines, and avoid any possible hold time problems on the device due to logic delay on the clock line. Intel FPGA devices also support dynamic PLL reconfiguration, which is the safest and most robust method of changing clock rates during device operation.

If your design has too many clocks to use the clock control block, or if dynamic reconfiguration is too complex for your design, you can implement a clock multiplexer in logic cells. However, if you use this implementation, consider simultaneous toggling inputs and ensure glitch-free transitions.

**Figure 31. Simple Clock Multiplexer in a 6-Input LUT**



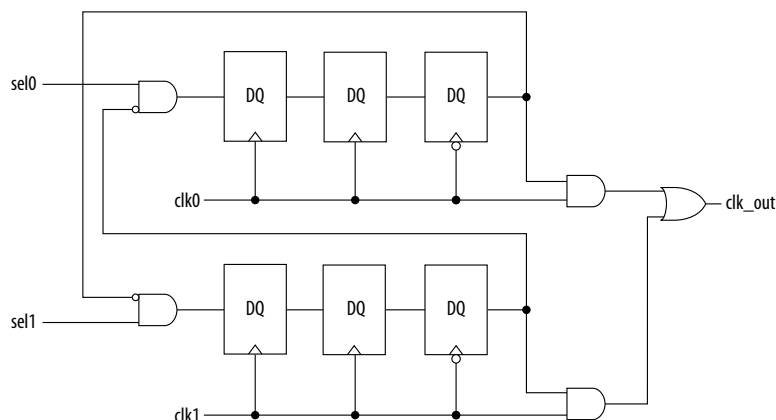
Each device datasheet describes how LUT outputs can glitch during a simultaneous toggle of input signals, independent of the LUT function. Even though the 4:1 MUX function does not generate detectable glitches during simultaneous data input toggles, some cell implementations of multiplexing logic exhibit significant glitches, so this



clock mux structure is not recommended. An additional problem with this implementation is that the output behaves erratically during a change in the `clk_select` signals. This behavior could create timing violations on all registers fed by the system clock and result in possible metastability.

A more sophisticated clock select structure can eliminate the simultaneous toggle and switching problems.

**Figure 32. Glitch-Free Clock Multiplexer Structure**



You can generalize this structure for any number of clock channels. The design ensures that no clock activates until all others are inactive for at least a few cycles, and that activation occurs while the clock is low. The design applies a `synthesis_keep` directive to the AND gates on the right side, which ensures there are no simultaneous toggles on the input of the `clk_out` OR gate.

**Note:** Switching from clock A to clock B requires that clock A continue to operate for at least a few cycles. If clock A stops immediately, the design sticks. The select signals are implemented as a “one-hot” control in this example, but you can use other encoding if you prefer. The input side logic is asynchronous and is not critical. This design can tolerate extreme glitching during the switch process.

#### Example 48. Verilog HDL Clock Multiplexing Design to Avoid Glitches

This example works with Verilog-2001.

```

module clock_mux (clk,clk_select,clk_out);

    parameter num_clocks = 4;

    input [num_clocks-1:0] clk;
    input [num_clocks-1:0] clk_select; // one hot
    output clk_out;

    genvar i;

    reg [num_clocks-1:0] ena_r0;
    reg [num_clocks-1:0] ena_r1;
    reg [num_clocks-1:0] ena_r2;
    wire [num_clocks-1:0] qualified_sel;

    // A look-up-table (LUT) can glitch when multiple inputs
    // change simultaneously. Use the keep attribute to
    // insert a hard logic cell buffer and prevent
    // the unrelated clocks from appearing on the same LUT.

```

```

wire [num_clocks-1:0] gated_clks /* synthesis keep */;

initial begin
    ena_r0 = 0;
    ena_r1 = 0;
    ena_r2 = 0;
end

generate
    for (i=0; i<num_clocks; i=i+1)
        begin : lp0
            wire [num_clocks-1:0] tmp_mask;
            assign tmp_mask = {num_clocks{1'b1}} ^ (1 << i);

            assign qualified_sel[i] = clk_select[i] & (~|(ena_r2 & tmp_mask));

            always @(posedge clk[i]) begin
                ena_r0[i] <= qualified_sel[i];
                ena_r1[i] <= ena_r0[i];
            end

            always @(negedge clk[i]) begin
                ena_r2[i] <= ena_r1[i];
            end

            assign gated_clks[i] = clk[i] & ena_r2[i];
        end
    endgenerate

    // These will not exhibit simultaneous toggle by construction
    assign clk_out = |gated_clks;

endmodule

```

### Related Links

[Intel FPGA IP Core Literature](#)

## 4.6.3 Adder Trees

Structuring adder trees appropriately to match your targeted Intel FPGA device architecture can provide significant improvements in your design's efficiency and performance.

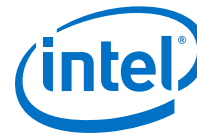
A good example of an application using a large adder tree is a finite impulse response (FIR) correlator. Using a pipelined binary or ternary adder tree appropriately can greatly improve the quality of your results.

### 4.6.3.1 Architectures with 6-Input LUTs in Adaptive Logic Modules

In Intel FPGA device families with 6-input LUT in their basic logic structure, ALMs can simultaneously add three bits. Take advantage of this feature by restructuring your code for better performance.

Although code targeting 4-input LUT architectures compiles successfully for 6-input LUT devices, the implementation can be inefficient. For example, to take advantage of the 6-input adaptive ALUT, you must rewrite large pipelined binary adder trees designed for 4-input LUT architectures. By restructuring the tree as a ternary tree, the design becomes much more efficient, significantly improving density utilization.





### Example 49. Verilog HDL Pipelined Ternary Tree

The example shows a pipelined adder, but partitioning your addition operations can help you achieve better results in non-pipelined adders as well. If your design is not pipelined, a ternary tree provides much better performance than a binary tree. For example, depending on your synthesis tool, the HDL code  $sum = (A + B + C) + (D + E)$  is more likely to create the optimal implementation of a 3-input adder for  $A + B + C$  followed by a 3-input adder for  $sum1 + D + E$  than the code without the parentheses. If you do not add the parentheses, the synthesis tool may partition the addition in a way that is not optimal for the architecture.

```

module ternary_adder_tree (a, b, c, d, e, clk, out);
    parameter width = 16;
    input [width-1:0] a, b, c, d, e;
    input    clk;
    output [width-1:0] out;

    wire [width-1:0] sum1, sum2;
    reg [width-1:0] sumreg1, sumreg2;
    // registers

    always @ (posedge clk)
        begin
            sumreg1 <= sum1;
            sumreg2 <= sum2;
        end

    // 3-bit additions
    assign sum1 = a + b + c;
    assign sum2 = sumreg1 + d + e;
    assign out = sumreg2;
endmodule

```

### 4.6.4 State Machine HDL Guidelines

Synthesis tools can recognize and encode Verilog HDL and VHDL state machines during synthesis. This section presents guidelines to secure the best results when you use state machines.

When a synthesis tool recognizes a piece of code as a state machine, it can implement techniques that improve the design area and performance. For example, the tool can recode the state variables to improve the quality of results, or use the known properties of state machines to optimize other parts of the design.

To achieve the best results, synthesis tools often use one-hot encoding for FPGA devices and minimal-bit encoding for CPLD devices, although the choice of implementation can vary for different state machines and different devices. Refer to your synthesis tool documentation for specific ways to control the manner in which state machines are encoded.

To ensure proper recognition and inference of state machines and to improve the quality of results, observe the following guidelines for both Verilog HDL and VHDL:

- Assign default values to outputs derived from the state machine so that synthesis does not generate unwanted latches.
- Separate the state machine logic from all arithmetic functions and datapaths, including assigning output values.
- If your design contains an operation that more than one state uses, define the operation outside the state machine and cause the output logic of the state machine to use this value.
- Use a simple asynchronous or synchronous `reset` to ensure a defined power-up state. If your state machine design contains more elaborate `reset` logic, such as both an asynchronous `reset` and an asynchronous load, the Intel Quartus Prime software generates regular logic rather than inferring a state machine.

If a state machine enters an illegal state due to a problem with the device, the design likely ceases to function correctly until the next `reset` of the state machine. Synthesis tools do not provide for this situation by default. The same issue applies to any other registers if there is some fault in the system. A `default` or `when others` clause does not affect this operation, assuming that your design never deliberately enters this state. Synthesis tools remove any logic generated by a default state if it is not reachable by normal state machine operation.

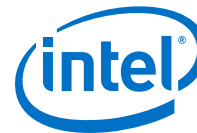
Many synthesis tools (including Intel Quartus Prime synthesis) have an option to implement a safe state machine. The Intel Quartus Prime software inserts extra logic to detect an illegal state and force the state machine's transition to the `reset` state. It is commonly used when the state machine can enter an illegal state. The most common cause of this situation is a state machine that has control inputs that come from another clock domain, such as the control logic for a dual-clock FIFO.

This option protects only state machines by forcing them into the `reset` state. All other registers in the design are not protected this way. If the design has asynchronous inputs, Intel recommends using a synchronization register chain instead of relying on the safe state machine option.

#### 4.6.4.1 Verilog HDL State Machines

To ensure proper recognition and inference of Verilog HDL state machines, observe the following additional Verilog HDL guidelines.

Refer to your synthesis tool documentation for specific coding recommendations. If the synthesis tool doesn't recognize and infer the state machine, the tool implements the state machine as regular logic gates and registers, and the state machine doesn't appear as a state machine in the **Analysis & Synthesis** section of the Intel Quartus Prime Compilation Report. In this case, Intel Quartus Prime synthesis does not perform any optimizations specific to state machines.



- If you are using the SystemVerilog standard, use enumerated types to describe state machines.
- Represent the states in a state machine with the parameter data types in Verilog-1995 and Verilog-2001, and use the parameters to make state assignments. This parameter implementation makes the state machine easier to read and reduces the risk of errors during coding.
- Do not directly use integer values for state variables, such as `next_state <= 0`. However, using an integer does not prevent inference in the Intel Quartus Prime software.
- Intel Quartus Prime software doesn't infer a state machine if the state transition logic uses arithmetic similar to the following example:

```

case (state)
  0: begin
    if (ena) next_state <= state + 2;
    else next_state <= state + 1;
    end
  1: begin
    ...
  endcase

```

- Intel Quartus Prime software doesn't infer a state machine if the state variable is an output.
- Intel Quartus Prime software doesn't infer a state machine for signed variables.

#### 4.6.4.1.1 Verilog-2001 State Machine Coding Example

The following module `verilog_fsm` is an example of a typical Verilog HDL state machine implementation. This state machine has five states.

The asynchronous reset sets the variable `state` to `state_0`. The sum of `in_1` and `in_2` is an output of the state machine in `state_1` and `state_2`. The difference (`in_1 - in_2`) is also used in `state_1` and `state_2`. The temporary variables `tmp_out_0` and `tmp_out_1` store the sum and the difference of `in_1` and `in_2`. Using these temporary variables in the various states of the state machine ensures proper resource sharing between the mutually exclusive states.

#### Example 50. Verilog-2001 State Machine

```

module verilog_fsm (clk, reset, in_1, in_2, out);
  input clk, reset;
  input [3:0] in_1, in_2;
  output [4:0] out;
  parameter state_0 = 3'b000;
  parameter state_1 = 3'b001;
  parameter state_2 = 3'b010;
  parameter state_3 = 3'b011;
  parameter state_4 = 3'b100;

  reg [4:0] tmp_out_0, tmp_out_1, tmp_out_2;
  reg [2:0] state, next_state;

  always @ (posedge clk or posedge reset)
  begin
    if (reset)
      state <= state_0;
    else
      state <= next_state;
  end
  always @ (*)

```

```

begin
  tmp_out_0 = in_1 + in_2;
  tmp_out_1 = in_1 - in_2;
  case (state)
    state_0: begin
      tmp_out_2 = in_1 + 5'b00001;
      next_state = state_1;
    end
    state_1: begin
      if (in_1 < in_2) begin
        next_state = state_2;
        tmp_out_2 = tmp_out_0;
      end
      else begin
        next_state = state_3;
        tmp_out_2 = tmp_out_1;
      end
    end
    state_2: begin
      tmp_out_2 = tmp_out_0 - 5'b00001;
      next_state = state_3;
    end
    state_3: begin
      tmp_out_2 = tmp_out_1 + 5'b00001;
      next_state = state_0;
    end
    state_4:begin
      tmp_out_2 = in_2 + 5'b00001;
      next_state = state_0;
    end
    default:begin
      tmp_out_2 = 5'b00000;
      next_state = state_0;
    end
  endcase
  end
  assign out = tmp_out_2;
endmodule

```

You can achieve an equivalent implementation of this state machine by using ``define` instead of the parameter data type, as follows:

```

`define state_0 3'b000
`define state_1 3'b001
`define state_2 3'b010
`define state_3 3'b011
`define state_4 3'b100

```

In this case, you assign ``state_x` instead of `state_x` to `state` and `next_state`, for example:

```

next_state <= `state_3;

```

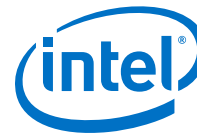
**Note:** Although Intel supports the ``define` construct, use the parameter data type, because it preserves the state names throughout synthesis.

#### 4.6.4.1.2 SystemVerilog State Machine Coding Example

Use the following coding style to describe state machines in SystemVerilog.

##### Example 51. SystemVerilog State Machine Using Enumerated Types

The module `enum_fsm` is an example of a SystemVerilog state machine implementation that uses enumerated types.



In Intel Quartus Prime Pro Edition synthesis, the enumerated type that defines the states for the state machine must be of an unsigned integer type. If you do not specify the enumerated type as `int unsigned`, synthesis uses a signed `int` type by default. In this case, the Intel Quartus Prime software synthesizes the design, but does not infer or optimize the logic as a state machine.

```

module enum_fsm (input clk, reset, input int data[3:0], output int o);
enum int unsigned { S0 = 0, S1 = 2, S2 = 4, S3 = 8 } state, next_state;
always_comb begin : next_state_logic
    next_state = S0;
    case(state)
        S0: next_state = S1;
        S1: next_state = S2;
        S2: next_state = S3;
        S3: next_state = S3;
    endcase
end
always_comb begin
    case(state)
        S0: o = data[3];
        S1: o = data[2];
        S2: o = data[1];
        S3: o = data[0];
    endcase
end
always_ff@(posedge clk or negedge reset) begin
    if(~reset)
        state <= S0;
    else
        state <= next_state;
    end
end
endmodule

```

#### 4.6.4.2 VHDL State Machines

To ensure proper recognition and inference of VHDL state machines, represent the different states with enumerated types, and use the corresponding types to make state assignments.

This implementation makes the state machine easier to read, and reduces the risk of errors during coding. If your RTL does not represent states with an enumerated type, Intel Quartus Prime synthesis (and other synthesis tools) do not recognize the state machine. Instead, synthesis implements the state machine as regular logic gates and registers. Consequently, the state machine does not appear in the state machine list of the Intel Quartus Prime Compilation Report, **Analysis & Synthesis** section. Moreover, Intel Quartus Prime synthesis does not perform any of the optimizations that are specific to state machines.

##### 4.6.4.2.1 VHDL State Machine Coding Example

The following state machine has five states. The asynchronous reset sets the variable `state` to `state_0`.

The sum of `in1` and `in2` is an output of the state machine in `state_1` and `state_2`. The difference (`in1 - in2`) is also used in `state_1` and `state_2`. The temporary variables `tmp_out_0` and `tmp_out_1` store the sum and the difference of `in1` and `in2`. Using these temporary variables in the various states of the state machine ensures proper resource sharing between the mutually exclusive states.



## Example 52. VHDL State Machine

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY vhdl_fsm IS
    PORT(
        clk: IN STD_LOGIC;
        reset: IN STD_LOGIC;
        in1: IN UNSIGNED(4 downto 0);
        in2: IN UNSIGNED(4 downto 0);
        out_1: OUT UNSIGNED(4 downto 0)
    );
END vhdl_fsm;
ARCHITECTURE rtl OF vhdl_fsm IS
    TYPE Tstate IS (state_0, state_1, state_2, state_3, state_4);
    SIGNAL state: Tstate;
    SIGNAL next_state: Tstate;
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF reset = '1' THEN
            state <= state_0;
        ELSIF rising_edge(clk) THEN
            state <= next_state;
        END IF;
    END PROCESS;
    PROCESS (state, in1, in2)
        VARIABLE tmp_out_0: UNSIGNED (4 downto 0);
        VARIABLE tmp_out_1: UNSIGNED (4 downto 0);
    BEGIN
        tmp_out_0 := in1 + in2;
        tmp_out_1 := in1 - in2;
        CASE state IS
            WHEN state_0 =>
                out_1 <= in1;
                next_state <= state_1;
            WHEN state_1 =>
                IF (in1 < in2) then
                    next_state <= state_2;
                    out_1 <= tmp_out_0;
                ELSE
                    next_state <= state_3;
                    out_1 <= tmp_out_1;
                END IF;
            WHEN state_2 =>
                IF (in1 < "0100") then
                    out_1 <= tmp_out_0;
                ELSE
                    out_1 <= tmp_out_1;
                END IF;
                next_state <= state_3;
            WHEN state_3 =>
                out_1 <= "11111";
                next_state <= state_4;
            WHEN state_4 =>
                out_1 <= in2;
                next_state <= state_0;
            WHEN OTHERS =>
                out_1 <= "00000";
                next_state <= state_0;
        END CASE;
    END PROCESS;
END rtl;
```



## 4.6.5 Multiplexer HDL Guidelines

Multiplexers form a large portion of the logic utilization in many FPGA designs. By optimizing your multiplexer logic, you ensure the most efficient implementation.

This section addresses common problems and provides design guidelines to achieve optimal resource utilization for multiplexer designs. The section also describes various types of multiplexers, and how they are implemented.

For more information, refer to the *Advanced Synthesis Cookbook*.

### 4.6.5.1 Intel Quartus Prime Software Option for Multiplexer Restructuring

Intel Quartus Prime Pro Edition synthesis provides the **Restructure Multiplexers** logic option that extracts and optimizes buses of multiplexers during synthesis. The default **Auto** for this option setting uses the optimization whenever beneficial for your design. You can turn the option on or off specifically to have more control over use.

Even with this Intel Quartus Prime-specific option turned on, it is beneficial to understand how your coding style can be interpreted by your synthesis tool, and avoid the situations that can cause problems in your design.

### 4.6.5.2 Multiplexer Types

This section addresses how Intel Quartus Prime synthesis creates multiplexers from various types of HDL code.

State machines, `CASE` statements, and `IF` statements are all common sources of multiplexer logic in designs. These HDL structures create different types of multiplexers, including binary multiplexers, selector multiplexers, and priority multiplexers.

The first step toward optimizing multiplexer structures for best results is to understand how Intel Quartus Prime infers and implements multiplexers from HDL code.

#### 4.6.5.2.1 Binary Multiplexers

Binary multiplexers select inputs based on binary-encoded selection bits.

Device families featuring 6-input look up tables (LUTs) are perfectly suited for 4:1 multiplexer building blocks (4 data and 2 select inputs). The extended input mode facilitates implementing 8:1 blocks, and the fractured mode handles residual 2:1 multiplexer pairs.

#### Example 53. Verilog HDL Binary-Encoded Multiplexers

```
case (sel)
  2'b00: z = a;
  2'b01: z = b;
  2'b10: z = c;
  2'b11: z = d;
endcase
```

#### 4.6.5.2.2 Selector Multiplexers

Selector multiplexers have a separate select line for each data input. The select lines for the multiplexer are one-hot encoded. Intel Quartus Prime commonly builds selector multiplexers as a tree of AND and OR gates.

Even though the implementation of a tree-shaped, N-input selector multiplexer is slightly less efficient than a binary multiplexer, in many cases the select signal is the output of a decoder. Intel Quartus Prime synthesis combines the selector and decoder into a binary multiplexer.

#### Example 54. Verilog HDL One-Hot-Encoded CASE Statement

```

case (sel)
  4'b0001: z = a;
  4'b0010: z = b;
  4'b0100: z = c;
  4'b1000: z = d;
  default: z = 1'bx;
endcase

```

#### 4.6.5.2.3 Priority Multiplexers

In priority multiplexers, the select logic implies a priority. The options to select the correct item must be checked in a specific order based on signal priority.

Synthesis tools commonly infer these structures from IF, ELSE, WHEN, SELECT, and ?: statements in VHDL or Verilog HDL.

#### Example 55. VHDL IF Statement Implying Priority

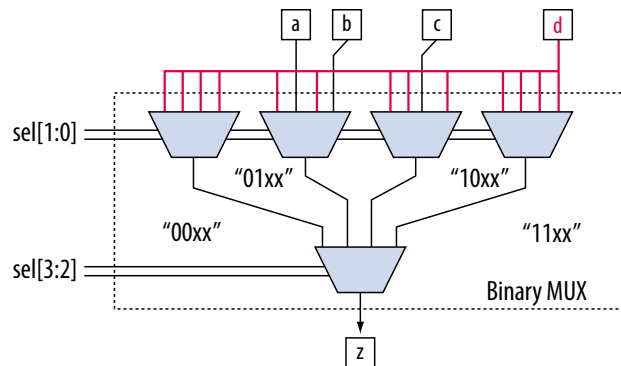
The multiplexers form a chain, evaluating each condition or select bit sequentially.

```

IF cond1 THEN z <= a;
ELSIF cond2 THEN z <= b;
ELSIF cond3 THEN z <= c;
ELSE z <= d;
END IF;

```

Figure 33. Priority Multiplexer Implementation of an IF Statement



Depending on the number of multiplexers in the chain, the timing delay through this chain can become large, especially for device families with 4-input LUTs.





To improve the timing delay through the multiplexer, avoid priority multiplexers if priority is not required. If the order of the choices is not important to the design, use a `CASE` statement to implement a binary or selector multiplexer instead of a priority multiplexer. If delay through the structure is important in a multiplexed design requiring priority, consider recoding the design to reduce the number of logic levels to minimize delay, especially along your critical paths.

#### 4.6.5.3 Implicit Defaults in `IF` Statements

The `IF` statements in Verilog HDL and VHDL can be a convenient way to specify conditions that do not easily lend themselves to a `CASE`-type approach.

However, using `IF` statements can result in complicated multiplexer trees that are not easy for synthesis tools to optimize. In particular, every `IF` statement has an implicit `ELSE` condition, even when it is not specified. These implicit defaults can cause additional complexity in a multiplexed design.

There are several ways you can simplify multiplexed logic and remove unneeded defaults. The optimal method may be to recode the design so the logic takes the structure of a 4:1 `CASE` statement. Alternatively, if priority is important, you can restructure the code to reduce default cases and flatten the multiplexer. Examine whether the default "`ELSE IF`" conditions are don't care cases. You may be able to create a default `ELSE` statement to make the behavior explicit. Avoid unnecessary default conditions in your multiplexer logic to reduce the complexity and logic utilization required to implement your design.

#### 4.6.5.4 `default` or `OTHERS` `CASE` Assignment

To fully specify the cases in a `CASE` statement, include a `default` (Verilog HDL) or `OTHERS` (VHDL) assignment.

This assignment is especially important in one-hot encoding schemes where many combinations of the select lines are unused. Specifying a case for the unused select line combinations gives the synthesis tool information about how to synthesize these cases, and is required by the Verilog HDL and VHDL language specifications.

For some designs you do not need to consider the outcome in the unused cases, because these cases are unreachable. For these types of designs, you can specify any value for the `default` or `OTHERS` assignment. However, the assignment value you choose can have a large effect on the logic utilization required to implement the design.

To obtain best results, explicitly define invalid `CASE` selections with a separate `default` or `OTHERS` statement, instead of combining the invalid cases with one of the defined cases.

If the value in the invalid cases is not important, specify those cases explicitly by assigning the `x` (don't care) logic value instead of choosing another value. This assignment allows your synthesis tool to perform the best area optimizations.

#### 4.6.6 Cyclic Redundancy Check Functions

CRC computations are used heavily by communications protocols and storage devices to detect any corruption of data. These functions are highly effective; there is a very low probability that corrupted data can pass a 32-bit CRC check



CRC functions typically use wide XOR gates to compare the data. The way synthesis tools flatten and factor these XOR gates to implement the logic in FPGA LUTs can greatly impact the area and performance results for the design. XOR gates have a cancellation property that creates an exceptionally large number of reasonable factoring combinations, so synthesis tools cannot always choose the best result by default.

The 6-input ALUT has a significant advantage over 4-input LUTs for these designs. When properly synthesized, CRC processing designs can run at high speeds in devices with 6-input ALUTs.

The following guidelines help you improve the quality of results for CRC designs in Intel FPGA devices.

#### 4.6.6.1 If Performance is Important, Optimize for Speed

To minimize area and depth of levels of logic, synthesis tools flatten XOR gates.

By default, Intel Quartus Prime Pro Edition synthesis targets area optimization for XOR gates. Therefore, for more focus on depth reduction, set the synthesis optimization technique to speed.

*Note:* Flattening for depth sometimes causes a significant increase in area.

#### 4.6.6.2 Use Separate CRC Blocks Instead of Cascaded Stages

Some designs optimize CRC to use cascaded stages (for example, four stages of 8 bits). In such designs, Intel Quartus Prime synthesis uses intermediate calculations (such as the calculations after 8, 24, or 32 bits) depending on the data width.

This design is not optimal for FPGA devices. The XOR cancellations that Intel Quartus Prime synthesis performs in CRC designs mean that the function does not require all the intermediate calculations to determine the final result. Therefore, forcing the use of intermediate calculations increases the area required to implement the function, as well as increasing the logic depth because of the cascading. It is typically better to create full separate CRC blocks for each data width that you require in the design, and then multiplex them together to choose the appropriate mode at a given time.

#### 4.6.6.3 Use Separate CRC Blocks Instead of Allowing Blocks to Merge

Synthesis tools often attempt to optimize CRC designs by sharing resources and extracting duplicates in two different CRC blocks because of the factoring options in the XOR logic.

The CRC logic allows significant reductions, but this works best when each CRC function is optimized separately. Check for duplicate extraction behavior if you have different CRC functions that are driven by common data signals or that feed the same destination signals.



If you are having problems with the quality of results and you see that two CRC functions are sharing logic, ensure that the blocks are synthesized independently using one of the following methods:

- Define each CRC block as a separate design partition in an hierarchical compilation design flow.
- Synthesize each CRC block as a separate project in your third-party synthesis tool and then write a separate Verilog Quartus Mapping (**.vqm**) or EDIF netlist file for each.

#### 4.6.6.4 Take Advantage of Latency if Available

If your design can use more than one cycle to implement the CRC functionality, adding registers and retiming the design can help reduce area, improve performance, and reduce power utilization.

If your synthesis tool offers a retiming feature (such as the Intel Quartus Prime software **Perform gate-level register retiming** option), you can insert an extra bank of registers at the input and allow the retiming feature to move the registers for better results. You can also build the CRC unit half as wide and alternate between halves of the data in each clock cycle.

#### 4.6.6.5 Save Power by Disabling CRC Blocks When Not in Use

CRC designs are heavy consumers of dynamic power because the logic toggles whenever there is a change in the design.

To save power, use clock enables to disable the CRC function for every clock cycle that the logic is not required. Some designs don't check the CRC results for a few clock cycles while other logic is performing. It is valuable to disable the CRC function even for this short amount of time.

#### 4.6.6.6 Use the Device Synchronous Load (sload) Signal to Initialize

The data in many CRC designs must be initialized to 1's before operation. If your target device supports the use of the `sload` signal, use it to set all the registers in your design to 1's before operation.

To enable use of the `sload` signal, follow the coding guidelines in this chapter. You can check the register equations in the Chip Planner to ensure that the signal was used as expected.

If you must force a register implementation using an `sload` signal, refer to *Designing with Low-Level Primitives User Guide* to see how you can use low-level device primitives.

#### Related Links

- [Secondary Register Control Signals Such as Clear and Clock Enable](#) on page 129
- [Designing with Low-Level Primitives User Guide](#)

### 4.6.7 Comparator HDL Guidelines

This section provides information about the different types of implementations available for comparators (<, >, or ==), and provides suggestions on how you can code your design to encourage a specific implementation. Synthesis tools, including Intel Quartus Prime Pro Edition synthesis, use device and context-specific implementation rules, and select the best one for your design.

Synthesis tools implement the == comparator in general logic cells. Additionally, synthesis tools implement the < comparison either using the carry chain or general logic cells. In devices with 6-input ALUTs, the carry chain is capable of comparing up to three bits per cell. The carry chain implementation tends to be faster than the general logic on standalone benchmark test cases, but can result in lower performance when it is part of a larger design due to the increased restriction on the Fitter. The area requirement is similar for most input patterns. The synthesis tools select an appropriate implementation based on the input pattern.

If you are using Intel Quartus Prime synthesis, you can guide the tool by using specific coding styles. To select a carry chain implementation explicitly, rephrase your comparison in terms of addition. As a simple example, the following coding style allows the synthesis tool to select the implementation, which is most likely using general logic cells in modern device families:

```
wire [6:0] a,b;  
wire alb = a<b;
```

In the following coding style, the synthesis tool uses a carry chain (except for a few cases, such as when the chain is very short or the signals *a* and *b* minimize to the same signal):

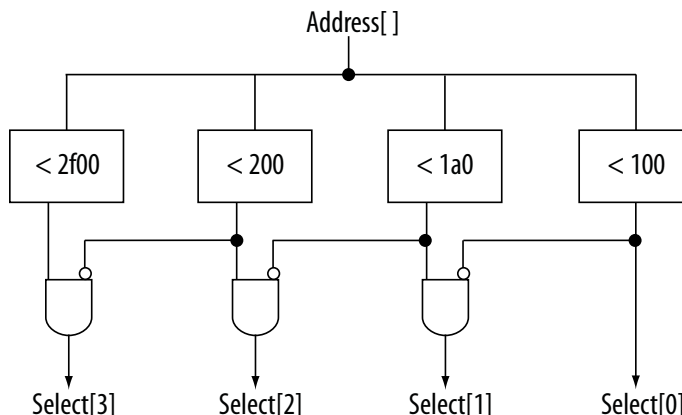
```
wire [6:0] a,b;  
wire [7:0] tmp = a - b;  
wire alb = tmp[7]
```

This second coding style uses the top bit of the *tmp* signal, which is 1 in twos complement logic if *a* is less than *b*, because the subtraction  $a - b$  results in a negative number.

If you have any information about the range of the input, you have “don’t care” values that you can use to optimize the design. Because this information is not available to the synthesis tool, you can often reduce the device area required to implement the comparator with specific hand implementation of the logic.

You can also check whether a bus value is within a constant range with a small amount of logic area by using the following logic structure . This type of logic occurs frequently in address decoders.

Figure 34. Example Logic Structure for Using Comparators to Check a Bus Value Range



### 4.6.8 Counter HDL Guidelines

The Intel Quartus Prime synthesis engine implements counters in HDL code as an adder followed by registers, and makes available register control signals such as enable (*ena*), synchronous clear (*sclr*), and synchronous load (*sload*). For best area utilization, ensure that the up and down control or controls are expressed in terms of one addition operator, instead of two separate addition operators.

If you use the following coding style, your synthesis engine may implement two separate carry chains for addition:

```
out <= count_up ? out + 1 : out - 1;
```

For simple designs, the synthesis engine identifies this coding style and optimizes the logic. However, in complex designs, or designs with preserve pragmas, the Compiler cannot optimize all logic, so more careful coding becomes necessary.

The following coding style requires only one adder along with some other logic:

```
out <= out + (count_up ? 1 : -1);
```

This style makes more efficient use of resources and area, since it uses only one carry chain adder, and the  $-1$  constant logic is implemented in the LUT before the adder.

## 4.7 Designing with Low-Level Primitives

Low-level HDL design is the practice of using low-level primitives and assignments to dictate a particular hardware implementation for a piece of logic. Low-level primitives are small architectural building blocks that assist you in creating your design.

With the Intel Quartus Prime software, you can use low-level HDL design techniques to force a specific hardware implementation that can help you achieve better resource utilization or faster timing results.

**Note:** Using low-level primitives is an optional advanced technique to help with specific design challenges. For many designs, synthesizing generic HDL source code and Intel FPGA IP cores give you the best results.



Low-level primitives allow you to use the following types of coding techniques:

- Instantiate the logic cell or `LCELL` primitive to prevent Intel Quartus Prime Pro Edition synthesis from performing optimizations across a logic cell
- Create carry and cascade chains using `CARRY`, `CARRY_SUM`, and `CASCADE` primitives
- Instantiate registers with specific control signals using `DFF` primitives
- Specify the creation of LUT functions by identifying the LUT boundaries
- Use I/O buffers to specify I/O standards, current strengths, and other I/O assignments
- Use I/O buffers to specify differential pin names in your HDL code, instead of using the automatically-generated negative pin name for each pair

For details about and examples of using these types of assignments, refer to the *Designing with Low-Level Primitives User Guide*.

#### Related Links

[Designing with Low-Level Primitives User Guide](#)

## 4.8 Document Revision History

The following revisions history applies to this chapter.

**Table 30. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"><li>• Described new <code>no_ram</code> synthesis attribute.</li></ul>
2017.05.08	17.0.0	<ul style="list-style-type: none"><li>• Updated example: Verilog HDL Multiply-Accumulator</li><li>• Updated information about use of safe state machine.</li><li>• Revised Check Read-During-Write Behavior.</li><li>• Revised Controlling RAM Inference and Implementation.</li><li>• Revised Single-Clock Synchronous RAM with Old Data Read-During-Write Behavior.</li><li>• Revised Single-Clock Synchronous RAM with New Data Read-During-Write Behavior.</li><li>• Updated and moved template for VHDL Single-Clock Simple Dual Port Synchronous RAM with New Data Read-During-Write Behavior.</li><li>• Revised Inferring ROM Functions from HDL Code.</li><li>• Removed example: VHDL 8-Bit Wide, 64-Bit Long Shift Register with Evenly Spaced Taps.</li><li>• Removed example: Verilog HDL D-Type Flipflop (Register) With <code>ena</code>, <code>aclr</code>, and <code>aload</code> Control Signals</li><li>• Removed example: VHDL D-Type Flipflop (Register) With <code>ena</code>, <code>aclr</code>, and <code>aload</code> Control Signals</li><li>• Added example: Verilog D-type Flipflop bus with Secondary Signals</li><li>• Removed references to 4-input LUT-based devices.</li><li>• Removed references to Integrated Synthesis.</li><li>• Created example: Avoid this VHDL Coding Style.</li></ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"><li>• Provided corrected Verilog HDL Pipelined Binary Tree and Ternary Tree examples.</li><li>• Implemented Intel rebranding.</li></ul>
2016.05.03	16.0.0	<ul style="list-style-type: none"><li>• Added information about use of safe state machine.</li><li>• Updated example code templates with latest coding styles.</li></ul>
<i>continued...</i>		



Date	Version	Changes
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li> </ul>
2015.05.04	15.0.0	Added information and reference about ramstyle attribute for sift register inference.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
2014.08.18	14.0.a10.0	<ul style="list-style-type: none"> <li>Added recommendation to use register pipelining to obtain high performance in DSP designs.</li> </ul>
2014.06.30	14.0.0	Removed obsolete MegaWizard Plug-In Manager support.
November 2013	13.1.0	Removed HardCopy device support.
June 2012	12.0.0	<ul style="list-style-type: none"> <li>Revised section on inserting Altera templates.</li> <li>Code update for Example 11-51.</li> <li>Minor corrections and updates.</li> </ul>
November 2011	11.1.0	<ul style="list-style-type: none"> <li>Updated document template.</li> <li>Minor updates and corrections.</li> </ul>
December 2010	10.1.0	<ul style="list-style-type: none"> <li>Changed to new document template.</li> <li>Updated Unintentional Latch Generation content.</li> <li>Code update for Example 11-18.</li> </ul>
July 2010	10.0.0	<ul style="list-style-type: none"> <li>Added support for mixed-width RAM</li> <li>Updated support for no_rw_check for inferring RAM blocks</li> <li>Added support for byte-enable</li> </ul>
November 2009	9.1.0	<ul style="list-style-type: none"> <li>Updated support for Controlling Inference and Implementation in Device RAM Blocks</li> <li>Updated support for Shift Registers</li> </ul>
March 2009	9.0.0	<ul style="list-style-type: none"> <li>Corrected and updated several examples</li> <li>Added support for Arria II GX devices</li> <li>Other minor changes to chapter</li> </ul>
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	<p>Updates for the Intel Quartus Prime software version 8.0 release, including:</p> <ul style="list-style-type: none"> <li>Added information to "RAM</li> <li>Functions—Inferring ALTSYNCRAM and ALTDPRAM Megafunctions from HDL Code" on page 6-13</li> <li>Added information to "Avoid Unsupported Reset and Control Conditions" on page 6-14</li> <li>Added information to "Check Read-During-Write Behavior" on page 6-16</li> <li>Added two new examples to "ROM Functions—Inferring ALTSYNCRAM and LPM_ROM Megafunctions from HDL Code" on page 6-28: Example 6-24 and Example 6-25</li> <li>Added new section: "Clock Multiplexing" on page 6-46</li> <li>Added hyperlinks to references within the chapter</li> <li>Minor editorial updates</li> </ul>

### Related Links

#### Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 5 Recommended Design Practices

---

This chapter provides design recommendations for Intel FPGA devices.

Current FPGA applications have reached the complexity and performance requirements of ASICs. In the development of complex system designs, design practices have an enormous impact on the timing performance, logic utilization, and system reliability of a device. Well-coded designs behave in a predictable and reliable manner even when retargeted to different families or speed grades. Good design practices also aid in successful design migration between FPGA and ASIC implementations for prototyping and production.

For optimal performance, reliability, and faster time-to-market when designing with Intel FPGA devices, you should adhere to the following guidelines:

- Understand the impact of synchronous design practices
- Follow recommended design techniques, including hierarchical design partitioning, and timing closure guidelines
- Take advantage of the architectural features in the targeted device

### 5.1 Following Synchronous FPGA Design Practices

The first step in good design methodology is to understand the implications of your design practices and techniques. This section outlines the benefits of optimal synchronous design practices and the hazards involved in other approaches.

Good synchronous design practices can help you meet your design goals consistently. Problems with other design techniques can include reliance on propagation delays in a device, which can lead to race conditions, incomplete timing analysis, and possible glitches.

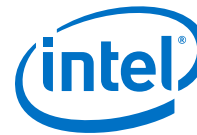
In a synchronous design, a clock signal triggers every event. If you ensure that all the timing requirements of the registers are met, a synchronous design behaves in a predictable and reliable manner for all process, voltage, and temperature (PVT) conditions. You can easily migrate synchronous designs to different device families or speed grades.

#### 5.1.1 Implementing Synchronous Designs

In a synchronous design, the clock signal controls the activities of all inputs and outputs.

On every active edge of the clock (usually the rising edge), the data inputs of registers are sampled and transferred to outputs. Following an active clock edge, the outputs of combinational logic feeding the data inputs of registers change values. This change triggers a period of instability due to propagation delays through the logic as the





signals go through several transitions and finally settle to new values. Changes that occur on data inputs of registers do not affect the values of their outputs until after the next active clock edge.

Because the internal circuitry of registers isolates data outputs from inputs, instability in the combinational logic does not affect the operation of the design if you meet the following timing requirements:

- Before an active clock edge, you must ensure that the data input has been stable for at least the setup time of the register.
- After an active clock edge, you must ensure that the data input remains stable for at least the hold time of the register.

When you specify all of your clock frequencies and other timing requirements, the Intel Quartus Prime Timing Analyzer reports actual hardware requirements for the setup times ( $t_{SU}$ ) and hold times ( $t_H$ ) for every pin in your design. By meeting these external pin requirements and following synchronous design techniques, you ensure that you satisfy the setup and hold times for all registers in your device.

*Tip:* To meet setup and hold time requirements on all input pins, any inputs to combinational logic that feed a register should have a synchronous relationship with the clock of the register. If signals are asynchronous, you can register the signals at the inputs of the device to help prevent a violation of the required setup and hold times.

When you violate the setup or hold time of a register, you might oscillate the output, or set the output to an intermediate voltage level between the high and low levels called a metastable state. In this unstable state, small perturbations such as noise in power rails can cause the register to assume either the high or low voltage level, resulting in an unpredictable valid state. Various undesirable effects can occur, including increased propagation delays and incorrect output states. In some cases, the output can even oscillate between the two valid states for a relatively long period of time.

### 5.1.2 Asynchronous Design Hazards

Some designers use asynchronous techniques such as ripple counters or pulse generators in programmable logic device (PLD) designs, enabling them to take “short cuts” to save device resources.

Asynchronous design techniques have inherent problems such as relying on propagation delays in a device, which can vary with temperature and voltage fluctuations, resulting in incomplete timing constraints and possible glitches and spikes.

Some asynchronous design structures depend on the relative propagation delays of signals to function correctly. In these cases, race conditions arise where the order of signal changes affect the output of the logic. Depending on how the design is placed and routed in the device, PLD designs can have varying timing delays with each compilation. Therefore, it is almost impossible to determine the timing delay associated with a particular block of logic ahead of time. As devices become faster due to process improvements, the delays in an asynchronous design may decrease, resulting in a design that does not function as expected. Relying on a particular delay also makes asynchronous designs difficult to migrate to different architectures, devices, or speed grades.

The timing of asynchronous design structures is often difficult or impossible to model with timing assignments and constraints. If you do not have complete or accurate timing constraints, the timing-driven algorithms used by your synthesis and place-and-route tools may not be able to perform the best optimizations, and the reported results may not be complete.

Some asynchronous design structures can generate harmful glitches, which are pulses that are very short compared to clock periods. Most glitches are generated by combinational logic. When the inputs to the combinational logic change, the outputs exhibit several glitches before they settle to their new values. These glitches can propagate through the combinational logic, leading to incorrect values on the outputs in asynchronous designs. In a synchronous design, glitches on the data inputs of registers are normal events that have no negative consequences because the data is not processed until the next clock edge.

## 5.2 HDL Design Guidelines

When designing with HDL code, you should understand how a synthesis tool interprets different HDL design techniques and what results to expect.

Your design style can affect logic utilization and timing performance, as well as the design's reliability. This section describes basic design techniques that ensure optimal synthesis results for designs targeted to Intel FPGA devices while avoiding several common causes of unreliability and instability. Intel recommends to design your combinational logic carefully to avoid potential problems. Pay attention to your clocking schemes so that you can maintain synchronous functionality and avoid timing problems.

### 5.2.1 Considerations for the Intel Hyperflex FPGA Architecture

The Intel Hyperflex FPGA architecture and the Hyper-Retimer require a review of the best design practices to achieve the highest clock rates possible.

While most common techniques of high-speed design apply to designing for the Intel Hyperflex architecture, you must use some new approaches to achieve the highest performance. Follow these general RTL design guidelines to enable the Hyper-Retimer to optimize design performance:

- Design in a way that facilitates register retiming by the Hyper-Retimer.
- Use a latency-insensitive design that supports the addition of pipeline stages at clock domain boundaries, top-level I/Os, and at the boundaries of functional blocks.
- Restructure RTL to avoid performance-limiting loops.

For more information about best design practices targeting Intel Stratix 10 devices, refer to the *Intel Stratix 10 High-Performance Design Handbook*.

#### Related Links

[RTL Design Guidelines](#)

In *Intel Stratix 10 High-Performance Design Handbook*

## 5.2.2 Optimizing Combinational Logic

Combinational logic structures consist of logic functions that depend only on the current state of the inputs. In Intel FPGAs, these functions are implemented in the look-up tables (LUTs) with either logic elements (LEs) or adaptive logic modules (ALMs).

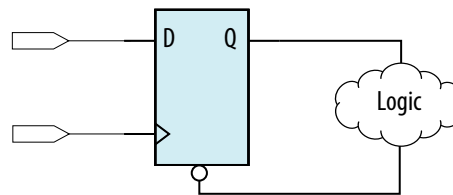
For cases where combinational logic feeds registers, the register control signals can implement part of the logic function to save LUT resources. By following the recommendations in this section, you can improve the reliability of your combinational design.

### 5.2.2.1 Avoid Combinational Loops

Combinational loops are among the most common causes of instability and unreliability in digital designs. Combinational loops generally violate synchronous design principles by establishing a direct feedback loop that contains no registers.

Avoid combinational loops whenever possible. In a synchronous design, feedback loops should include registers. For example, a combinational loop occurs when the left-hand side of an arithmetic expression also appears on the right-hand side in HDL code. A combinational loop also occurs when you feed back the output of a register to an asynchronous pin of the same register through combinational logic.

**Figure 35. Combinational Loop Through Asynchronous Control Pin**



*Tip:* Use recovery and removal analysis to perform timing analysis on asynchronous ports, such as `clear` or `reset` in the Intel Quartus Prime software.

Combinational loops are inherently high-risk design structures for the following reasons:

- Combinational loop behavior generally depends on relative propagation delays through the logic involved in the loop. As discussed, propagation delays can change, which means the behavior of the loop is unpredictable.
- In many design tools, combinational loops can cause endless computation loops. Most tools break open combinational loops to process the design. The various tools used in the design flow may open a given loop differently, and process it in a way inconsistent with the original design intent.

### 5.2.2.2 Avoid Unintended Latch Inference

Avoid using latches to ensure that you can completely analyze the timing performance and reliability of your design. A latch is a small circuit with combinational feedback that holds a value until a new value is assigned. You can implement latches with the Intel Quartus Prime Text Editor or Block Editor.



A common mistake in HDL code is unintended latch inference; Intel Quartus Prime Synthesis issues a warning message if this occurs. Unlike other technologies, a latch in FPGA architecture is not significantly smaller than a register. However, the architecture is not optimized for latch implementation and latches generally have slower timing performance compared to equivalent registered circuitry.

Latches have a transparent mode in which data flows continuously from input to output. A positive latch is in transparent mode when the enable signal is high (low for a negative latch). In transparent mode, glitches on the input can pass through to the output because of the direct path created. This presents significant complexity for timing analysis. Typical latch schemes use multiple enable phases to prevent long transparent paths from occurring. However, timing analysis cannot identify these safe applications.

The Timing Analyzer analyzes latches as synchronous elements clocked on the falling edge of the positive latch signal by default. It allows you to treat latches as having nontransparent start and end points. Be aware that even an instantaneous transition through transparent mode can lead to glitch propagation. The Timing Analyzer cannot perform cycle-borrowing analysis.

Due to various timing complexities, latches have limited support in formal verification tools. Therefore, you should not rely on formal verification for a design that includes latches.

#### Related Links

[Avoid Unintentional Latch Generation](#) on page 130

### 5.2.2.3 Avoid Delay Chains in Clock Paths

Delays in PLD designs can change with each placement and routing cycle. Effects such as rise and fall time differences and on-chip variation mean that delay chains, especially those placed on clock paths, can cause significant problems in your design. Avoid using delay chains to prevent these kinds of problems.

You require delay chains when you use two or more consecutive nodes with a single fan-in and a single fan-out to cause delay. Inverters are often chained together to add delay. Delay chains are sometimes used to resolve race conditions created by other asynchronous design practices.

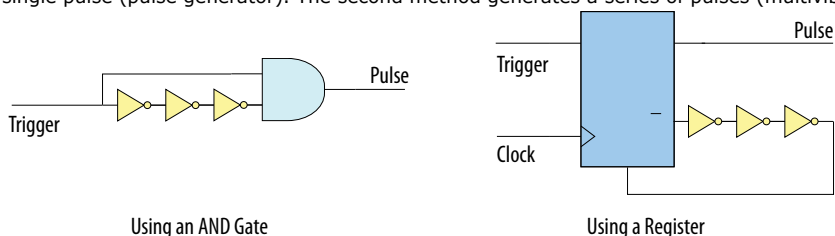
In some ASIC designs, delays are used for buffering signals as they are routed around the device. This functionality is not required in FPGA devices because the routing structure provides buffers throughout the device.

### 5.2.2.4 Use Synchronous Pulse Generators

Use synchronous techniques to design pulse generators.

**Figure 36. Asynchronous Pulse Generators**

The figure shows two methods for asynchronous pulse generation. The first method uses a delay chain to generate a single pulse (pulse generator). The second method generates a series of pulses (multivibrators).

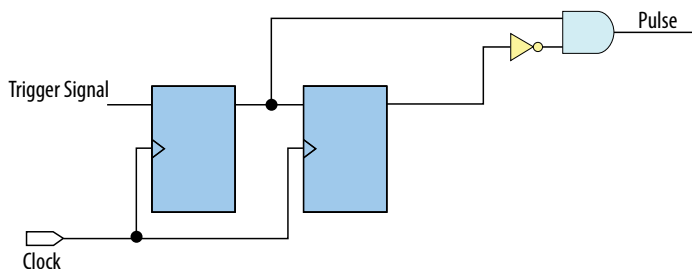


In the first method, a trigger signal feeds both inputs of a 2-input AND gate, and the design adds inverters to one of the inputs to create a delay chain. The width of the pulse depends on the time differences between the path that feeds the gate directly and the path that goes through the delay chain. This is the same mechanism responsible for the generation of glitches in combinational logic following a change of input values. This technique artificially increases the width of the glitch.

In the second method, a register’s output drives its asynchronous reset signal through a delay chain. The register resets itself asynchronously after a certain delay. The Compiler can determine the pulse width only after placement and routing, when routing and propagation delays are known. You cannot reliably create a specific pulse width when creating HDL code, and it cannot be set by EDA tools. The pulse may not be wide enough for the application under all PVT conditions. Also, the pulse width changes if you change to a different device. Additionally, verification is difficult because static timing analysis cannot verify the pulse width.

Multivibrators use a glitch generator to create pulses, together with a combinational loop that turns the circuit into an oscillator. This method creates additional problems because of the number of pulses involved. Additionally, when the structures generate multiple pulses, they also create a new artificial clock in the design that must be analyzed by design tools.

**Figure 37. Recommended Synchronous Pulse-Generation Technique**



The pulse width is always equal to the clock period. This pulse generator is predictable, can be verified with timing analysis, and is easily moved to other architectures, devices, or speed grades.

### 5.2.3 Optimizing Clocking Schemes

Like combinational logic, clocking schemes have a large effect on the performance and reliability of a design.

Avoid using internally generated clocks (other than PLLs) wherever possible because they can cause functional and timing problems in the design. Clocks generated with combinational logic can introduce glitches that create functional problems, and the delay inherent in combinational logic can lead to timing problems.

**Tip:** Specify all clock relationships in the Intel Quartus Prime software to allow for the best timing-driven optimizations during fitting and to allow correct timing analysis. Use clock setting assignments on any derived or internal clocks to specify their relationship to the base clock.

Use global device-wide, low-skew dedicated routing for all internally-generated clocks, instead of routing clocks on regular routing lines.

Avoid data transfers between different clocks wherever possible. If you require a data transfer between different clocks, use FIFO circuitry. You can use the clock uncertainty features in the Intel Quartus Prime software to compensate for the variable delays between clock domains. Consider setting a clock setup uncertainty and clock hold uncertainty value of 10% to 15% of the clock delay.

The following sections provide specific examples and recommendations for avoiding clocking scheme problems.

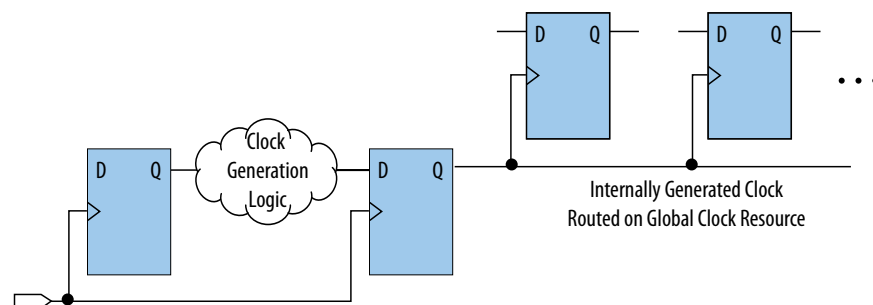
### 5.2.3.1 Register Combinational Logic Outputs

If you use the output from combinational logic as a clock signal or as an asynchronous reset signal, you can expect to see glitches in your design. In a synchronous design, glitches on data inputs of registers are normal events that have no consequences. However, a glitch or a spike on the clock input (or an asynchronous input) to a register can have significant consequences.

Narrow glitches can violate the register’s minimum pulse width requirements. Setup and hold requirements might also be violated if the data input of the register changes when a glitch reaches the clock input. Even if the design does not violate timing requirements, the register output can change value unexpectedly and cause functional hazards elsewhere in the design.

To avoid these problems, you should always register the output of combinational logic before you use it as a clock signal.

**Figure 38. Recommended Clock-Generation Technique**



Registering the output of combinational logic ensures that glitches generated by the combinational logic are blocked at the data input of the register.



### 5.2.3.2 Avoid Asynchronous Clock Division

Designs often require clocks that you create by dividing a master clock. Most Intel FPGAs provide dedicated phase-locked loop (PLL) circuitry for clock division. Using dedicated PLL circuitry can help you avoid many of the problems that can be introduced by asynchronous clock division logic.

When you must use logic to divide a master clock, always use synchronous counters or state machines. Additionally, create your design so that registers always directly generate divided clock signals, and route the clock on global clock resources. To avoid glitches, do not decode the outputs of a counter or a state machine to generate clock signals.

### 5.2.3.3 Avoid Ripple Counters

To simplify verification, avoid ripple counters in your design. In the past, FPGA designers implemented ripple counters to divide clocks by a power of two because the counters are easy to design and may use fewer gates than their synchronous counterparts.

Ripple counters use cascaded registers, in which the output pin of one register feeds the clock pin of the register in the next stage. This cascading can cause problems because the counter creates a ripple clock at each stage. These ripple clocks must be handled properly during timing analysis, which can be difficult and may require you to make complicated timing assignments in your synthesis and placement and routing tools.

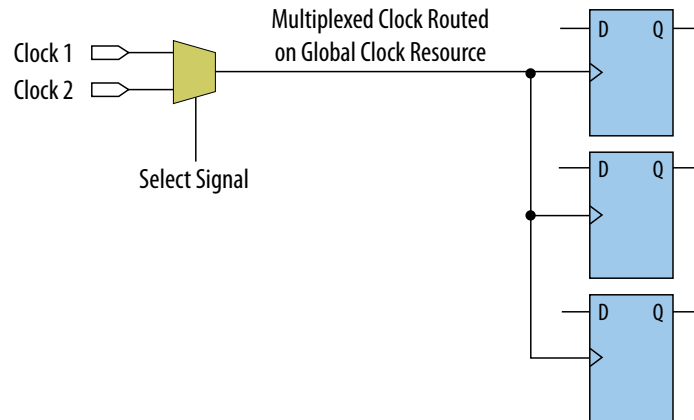
You can often use ripple clock structures to make ripple counters out of the smallest amount of logic possible. However, in all Intel devices supported by the Intel Quartus Prime software, using a ripple clock structure to reduce the amount of logic used for a counter is unnecessary because the device allows you to construct a counter using one logic element per counter bit. You should avoid using ripple counters completely.

### 5.2.3.4 Use Multiplexed Clocks

Use clock multiplexing to operate the same logic function with different clock sources. In these designs, multiplexing selects a clock source.

For example, telecommunications applications that deal with multiple frequency standards often use multiplexed clocks.

**Figure 39. Multiplexing Logic and Clock Sources**



Adding multiplexing logic to the clock signal can create the problems addressed in the previous sections, but requirements for multiplexed clocks vary widely, depending on the application. Clock multiplexing is acceptable when the clock signal uses global clock routing resources and if the following criteria are met:

- The clock multiplexing logic does not change after initial configuration
- The design uses multiplexing logic to select a clock for testing purposes
- Registers are always reset when the clock switches
- A temporarily incorrect response following clock switching has no negative consequences

If the design switches clocks in real time with no reset signal, and your design cannot tolerate a temporarily incorrect response, you must use a synchronous design so that there are no timing violations on the registers, no glitches on clock signals, and no race conditions or other logical problems. By default, the Intel Quartus Prime software optimizes and analyzes all possible paths through the multiplexer and between both internal clocks that may come from the multiplexer. This may lead to more restrictive analysis than required if the multiplexer is always selecting one particular clock. If you do not require the more complete analysis, you can assign the output of the multiplexer as a base clock in the Intel Quartus Prime software, so that all register-to-register paths are analyzed using that clock.

*Tip:* Use dedicated hardware to perform clock multiplexing when it is available, instead of using multiplexing logic. For example, you can use the clock-switchover feature or clock control block available in certain Intel FPGA devices. These dedicated hardware blocks ensure that you use global low-skew routing lines and avoid any possible hold time problems on the device due to logic delay on the clock line.

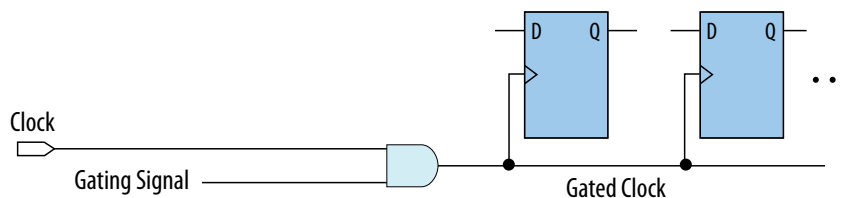
*Note:* For device-specific information about clocking structures, refer to the appropriate device data sheet or handbook on the Literature page of the Altera website.

### 5.2.3.5 Use Gated Clocks

Gated clocks turn a clock signal on and off using an enable signal that controls gating circuitry. When a clock is turned off, the corresponding clock domain is shut down and becomes functionally inactive.



**Figure 40. Gated Clock**



You can use gated clocks to reduce power consumption in some device architectures by effectively shutting down portions of a digital circuit when they are not in use. When a clock is gated, both the clock network and the registers driven by it stop toggling, thereby eliminating their contributions to power consumption. However, gated clocks are not part of a synchronous scheme and therefore can significantly increase the effort required for design implementation and verification. Gated clocks contribute to clock skew and make device migration difficult. These clocks are also sensitive to glitches, which can cause design failure.

Use dedicated hardware to perform clock gating rather than an AND or OR gate. For example, you can use the clock control block in newer Intel FPGA devices to shut down an entire clock network. Dedicated hardware blocks ensure that you use global routing with low skew, and avoid any possible hold time problems on the device due to logic delay on the clock line.

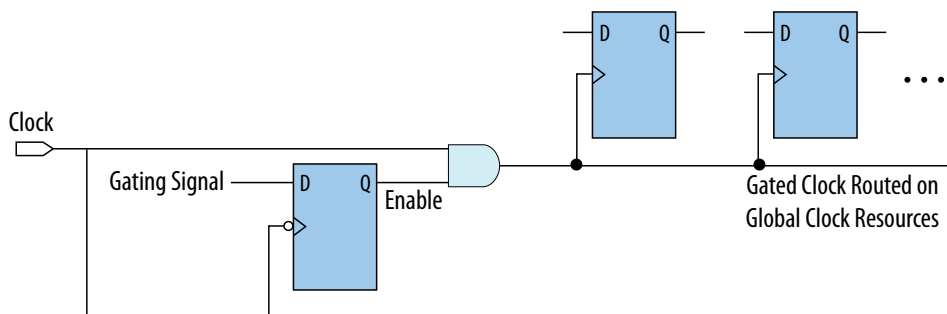
From a functional point of view, you can shut down a clock domain in a purely synchronous manner using a synchronous clock enable signal. However, when using a synchronous clock enable scheme, the clock network continues toggling. This practice does not reduce power consumption as much as gating the clock at the source does. In most cases, use a synchronous scheme.

### 5.2.3.5.1 Recommended Clock-Gating Methods

Use gated clocks only when your target application requires power reduction and gated clocks provide the required reduction in your device architecture. If you must use clocks gated by logic, follow a robust clock-gating methodology and ensure the gated clock signal uses dedicated global clock routing.

You can gate a clock signal at the source of the clock network, at each register, or somewhere in between. Since the clock network contributes to switching power consumption, gate the clock at the source whenever possible to shut down the entire clock network instead of further along.

**Figure 41. Recommended Clock-Gating Technique for Clock Active on Rising Edge**



To generate a gated clock with the recommended technique, use a register that triggers on the inactive edge of the clock. With this configuration, only one input of the gate changes at a time, preventing glitches or spikes on the output. If the clock is active on the rising edge, use an AND gate. Conversely, for a clock that is active on the falling edge, use an OR gate to gate the clock and register

Pay attention to the delay through the logic generating the enable signal, because the enable command must be ready in less than one-half the clock cycle. This might cause problems if the logic that generates the enable command is particularly complex, or if the duty cycle of the clock is severely unbalanced. However, careful management of the duty cycle and logic delay may be an acceptable solution when compared with problems created by other methods of gating clocks.

In the Timing Analyzer, ensure to apply a clock setting to the output of the AND gate. Otherwise, the timing analyzer might analyze the circuit using the clock path through the register as the longest clock path and the path that skips the register as the shortest clock path, resulting in artificial clock skew.

In certain cases, converting the gated clocks to clock enable pins may help reduce glitch and clock skew, and eventually produce a more accurate timing analysis. You can set the Intel Quartus Prime software to automatically convert gated clocks to clock enable pins by turning on the **Auto Gated Clock Conversion** option. The conversion applies to two types of gated clocking schemes: single-gated clock and cascaded-gated clock.

**Related Links**

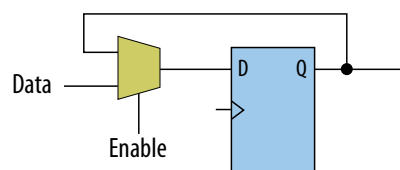
- [Advanced Synthesis Settings](#) on page 230
- [Auto Gated Clock Conversion logic option](#)  
In Intel Quartus Prime Help

**5.2.3.6 Use Synchronous Clock Enables**

To turn off a clock domain in a synchronous manner, use a synchronous clock enable signal. FPGAs efficiently support clock enable signals because there is a dedicated clock enable signal available on all device registers.

This scheme does not reduce power consumption as much as gating the clock at the source because the clock network keeps toggling, and performs the same function as a gated clock by disabling a set of registers. Insert a multiplexer in front of the data input of every register to either load new data, or copy the output of the register.

**Figure 42. Synchronous Clock Enable**



When designing for Intel Stratix 10 devices, consider that high fan-out clock enable signals can limit the performance achievable by the Hyper- Retimer. For specific recommendations, refer to the *Intel Stratix 10 High-Performance Design Handbook*.



## Related Links

[Clock Enable Strategies](#)

In *Intel Stratix 10 High-Performance Design Handbook*

## 5.2.4 Optimizing Physical Implementation and Timing Closure

This section provides design and timing closure techniques for high speed or complex core logic designs with challenging timing requirements. These techniques may also be helpful for low or medium speed designs.

### 5.2.4.1 Planning Physical Implementation

When planning a design, consider the following elements of physical implementation:

- The number of unique clock domains and their relationships
- The amount of logic in each functional block
- The location and direction of data flow between blocks
- How data routes to the functional blocks between I/O interfaces

Interface-wide control or status signals may have competing or opposing constraints. For example, when a functional block's control or status signals interface with physical channels from both sides of the device. In such cases you must provide enough pipeline register stages to allow these signals to traverse the width of the device. In addition, you can structure the hierarchy of the design into separate logic modules for each side of the device. The side modules can generate and use registered control signals per side. This simplifies floorplanning, particularly in designs with transceivers, by placing per-side logic near the transceivers.

When adding register stages to pipeline control signals, turn off **Auto Shift Register Replacement** in the **Assignment Editor (Assignments > Assignment Editor)** for each register as needed. By default, chains of registers can be converted to a RAM-based implementation based on performance and resource estimates. Since pipelining helps meet timing requirements over long distance, this assignment ensures that control signals are not converted.

### 5.2.4.2 Planning FPGA Resources

Your design requirements impact the use of FPGA resources. Plan functional blocks with appropriate global, regional, and dual-regional network signals in mind.

In general, after allocating the clocks in a design, use global networks for the highest fan-out control signals. When a global network signal distributes a high fan-out control signal, the global signal can drive logic anywhere in the device. Similarly, when using a regional network signal, the driven logic must be in one quadrant of the device, or half the device for a dual-regional network signal. Depending on data flow and physical locations of the data entry and exit between the I/Os and the device, restricting a functional block to a quadrant or half the device may not be practical for performance or resource requirements.

When floorplanning a design, consider the balance of different types of device resources, such as memory, logic, and DSP blocks in the main functional blocks. For example, if a design is memory intensive with a small amount of logic, it may be difficult to develop an effective floorplan. Logic that interfaces with the memory would



have to spread across the chip to access the memory. In this case, it is important to use enough register stages in the data and control paths to allow signals to traverse the chip to access the physically disparate resources needed.

### 5.2.4.3 Optimizing for Timing Closure

To achieve timing closure for your design, you can enable compilation settings in the Intel Quartus Prime software, or you can directly modify your timing constraints.

#### Compilation Settings for Timing Closure

**Note:** Changes in project settings can significantly increase compilation time. You can view the performance gain versus runtime cost by reviewing the Fitter messages after design processing.

**Table 31. Compilation Settings that Impact Timing Closure**

Setting	Location	Effect on Timing Closure
<b>Allow Register Duplication</b>	<b>Assignments &gt; Settings &gt; Compiler Settings &gt; Advanced Settings (Fitter)</b>	This technique is most useful where registers have high fan-out, or where the fan-out is in physically distant areas of the device.  Review the netlist optimizations report and consider manually duplicating registers automatically added by physical synthesis. You can also locate the original and duplicate registers in the Chip Planner. Compare their locations, and if the fan-out is improved, modify the code and turn off register duplication to save compile time.
<b>Prevent Register Retiming</b>	<b>Assignments &gt; Settings &gt; Compiler Settings &gt; Advanced Settings (Fitter)</b>	Useful if some combinatorial paths between registers exceed the timing goal while other paths fall short.  If a design is already heavily pipelined, register retiming is less likely to provide significant performance gains, since there should not be significantly unbalanced levels of logic across pipeline stages.



### Guidelines for Optimizing Timing Closure using Timing Constraints

Appropriate timing constraints are essential to achieving timing closure. Use the following general guidelines in applying timing constraints:

- Apply multicycle constraints in your design wherever single-cycle timing analysis is not necessary.
- Apply False Path constraints to all asynchronous clock domain crossings or resets in the design. This technique prevents overconstraining and the Fitter focuses only on critical paths to reduce compile time. However, overconstraining timing critical clock domains can sometimes provide better timing results and lower compile times than physical synthesis.
- Overconstrain rather than using physical synthesis when the slack improvement from physical synthesis is near zero. Overconstrain the frequency requirement on timing critical clock domains by using setup uncertainty.
- When evaluating the effect of constraint changes on performance and runtime, compile the design with at least three different seeds to determine the average performance and runtime effects. Different constraint combinations produce various results. Three samples or more establish a performance trend. Modify your constraints based on performance improvement or decline.
- Leave settings at the default value whenever possible. Increasing performance constraints can increase the compile time significantly. While those increases may be necessary to close timing on a design, using the default settings whenever possible minimizes compile time.

#### Related Links

[Design Evaluation for Timing Closure](#)

*In Intel Quartus Prime Pro Edition Handbook Volume 2*

### 5.2.4.4 Optimizing Critical Timing Paths

To close timing in high speed designs, review paths with the largest timing failures. Correcting a single, large timing failure can result in a very significant timing improvement.

Review the register placement and routing paths by clicking **Tools > Chip Planner**. Large timing failures on high fan-out control signals can be caused by any of the following conditions:

- Sub-optimal use of global networks
- Signals that traverse the chip on local routing without pipelining
- Failure to correct high fan-out by register duplication

For high-speed and high-bandwidth designs, optimize speed by reducing bus width and wire usage. To reduce wire usage, move the data as little as possible. For example, if a block of logic functions on a few bits of a word, store inactive bits in a FIFO or memory. Memory is cheaper and denser than registers, and reduces wire usage.

#### Related Links

[Exploring Paths in the Chip Planner](#)

*In Intel Quartus Prime Pro Edition Handbook Volume 2*

## 5.2.5 Optimizing Power Consumption

The total FPGA power consumption is comprised of I/O power, core static power, and core dynamic power. Knowledge of the relationship between these components is fundamental in calculating the overall total power consumption.

You can use various optimization techniques and tools to minimize power consumption when applied during FPGA design implementation. The Intel Quartus Prime software offers power-driven compilation features to fully optimize device power consumption. Power-driven compilation focuses on reducing your design's total power consumption using power-driven synthesis and power-driven placement and routing.

### Related Links

#### [Power Optimization](#)

In *Intel Quartus Prime Pro Edition Handbook Volume 2*

## 5.2.6 Managing Design Metastability

In FPGA designs, synchronization of asynchronous signals can cause metastability. You can use the Intel Quartus Prime software to analyze the mean time between failures (MTBF) due to metastability. A high metastability MTBF indicates a more robust design.

### Related Links

- [Managing Metastability with the Intel Quartus Prime Software](#) on page 986
- [Metastability Analysis and Optimization Techniques](#)  
In *Intel Quartus Prime Pro Edition Handbook Volume 2*

## 5.3 Use Clock and Register-Control Architectural Features

In addition to following general design guidelines, you must code your design with the device architecture in mind. FPGAs provide device-wide clocks and register control signals that can improve performance.

### 5.3.1 Use Global Reset Resources

ASIC designs may use local resets to avoid long routing delays. Take advantage of the device-wide asynchronous reset pin available on most FPGAs to eliminate these problems. This reset signal provides low-skew routing across the device.

The following are three types of resets used in synchronous circuits:

- Synchronous Reset
- Asynchronous Reset
- Synchronized Asynchronous Reset—preferred when designing an FPGA circuit

#### 5.3.1.1 Use Synchronous Resets

The synchronous reset ensures that the circuit is fully synchronous. You can easily time the circuit with the Intel Quartus Prime Timing Analyzer.



Because clocks that are synchronous to each other launch and latch the reset signal, the data arrival and data required times are easily determined for proper slack analysis. The synchronous reset is easier to use with cycle-based simulators.

There are two methods by which a reset signal can reach a register; either by being gated in with the data input, or by using an LAB-wide control signal (`syncclr`). If you use the first method, you risk adding an additional gate delay to the circuit to accommodate the reset signal, which causes increased data arrival times and negatively impacts setup slack. The second method relies on dedicated routing in the LAB to each register, but this is slower than an asynchronous reset to the same register.

**Figure 43. Synchronous Reset**

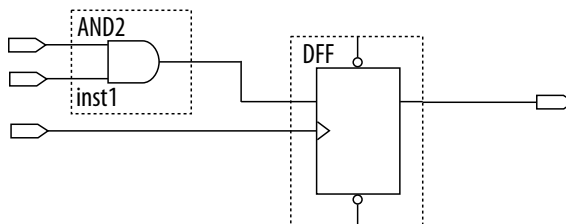
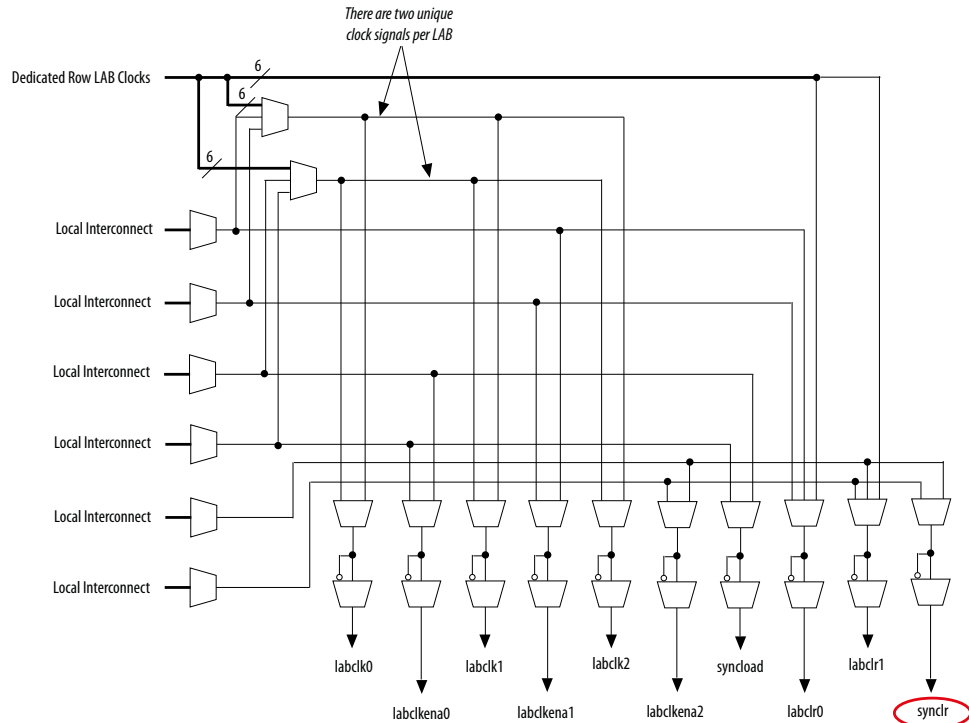
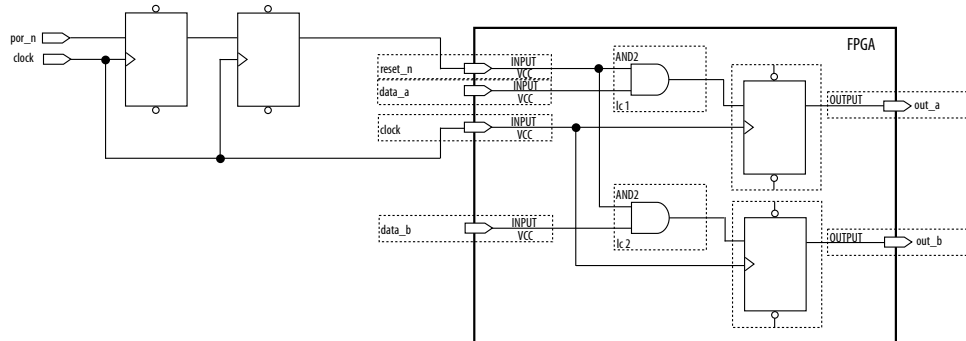


Figure 44. LAB-Wide Control Signals



Consider two types of synchronous resets when you examine the timing analysis of synchronous resets—externally synchronized resets and internally synchronized resets. Externally synchronized resets are synchronized to the clock domain outside the FPGA, and are not very common. A power-on asynchronous reset is dual-rank synchronized externally to the system clock and then brought into the FPGA. Inside the FPGA, gate this reset with the data input to the registers to implement a synchronous reset.

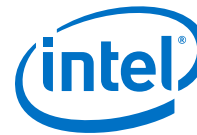
Figure 45. Externally Synchronized Reset



The following example shows the Verilog HDL equivalent of the schematic. When you use synchronous resets, the reset signal is not put in the sensitivity list.

The following example shows the necessary modifications that you should make to the internally synchronized reset.





### Example 56. Verilog HDL Code for Externally Synchronized Reset

```

module sync_reset_ext (
    input  clock,
    input  reset_n,
    input  data_a,
    input  data_b,
    output out_a,
    output out_b
);
reg    reg1, reg2;
assign out_a = reg1;
assign out_b = reg2;
always @ (posedge clock)
begin
    if (!reset_n)
    begin
        reg1    <= 1'b0;
        reg2    <= 1'b0;
    end
    else
    begin
        reg1    <= data_a;
        reg2    <= data_b;
    end
end
endmodule // sync_reset_ext

```

The following example shows the constraints for the externally synchronous reset. Because the external reset is synchronous, you only need to constrain the `reset_n` signal as a normal input signal with `set_input_delay` constraint for `-max` and `-min`.

### Example 57. SDC Constraints for Externally Synchronized Reset

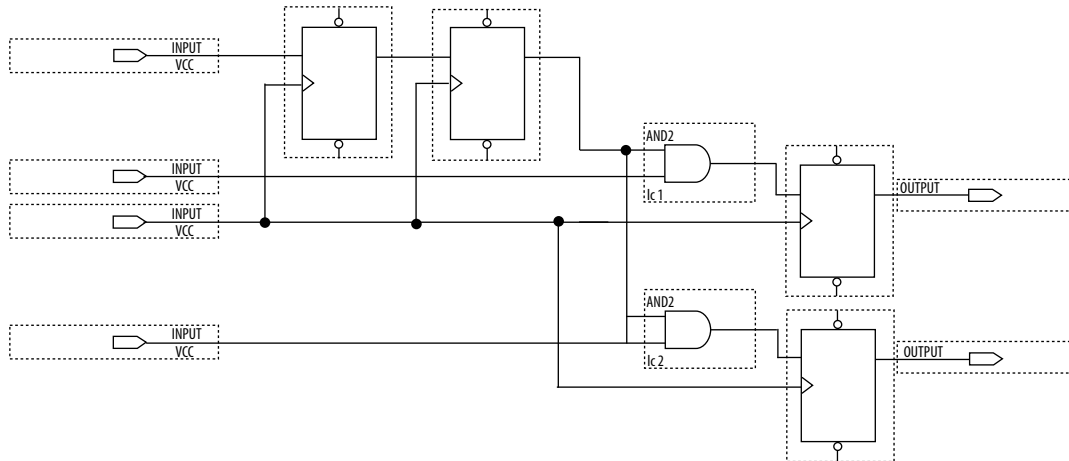
```

# Input clock - 100 MHz
create_clock [get_ports {clock}] \
    -name {clock} \
    -period 10.0 \
    -waveform {0.0 5.0}
# Input constraints on low-active reset
# and data
set_input_delay 7.0 \
    -max \
    -clock [get_clocks {clock}] \
    [get_ports {reset_n data_a data_b}]
set_input_delay 1.0 \
    -min \
    -clock [get_clocks {clock}] \
    [get_ports {reset_n data_a data_b}]

```

More often, resets coming into the device are asynchronous, and must be synchronized internally before being sent to the registers.

Figure 46. Internally Synchronized Reset



The following example shows the Verilog HDL equivalent of the schematic. Only the clock edge is in the sensitivity list for a synchronous reset.

Example 58. Verilog HDL Code for Internally Synchronized Reset

```

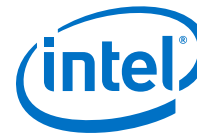
module sync_reset (
    input clock,
    input reset_n,
    input data_a,
    input data_b,
    output out_a,
    output out_b,
    output rst_n
);
    reg reg1, reg2;
    reg reg3, reg4;

    assign out_a = reg1;
    assign out_b = reg2;
    assign rst_n = reg4;

    always @ (posedge clock)
    begin
        if (!rst_n)
            begin
                reg1 <= 1'b0;
                reg2 <= 1'b0;
            end
        else
            begin
                reg1 <= data_a;
                reg2 <= data_b;
            end
        end
    end

    always @ (posedge clock)
    begin
        reg3 <= reset_n;
        reg4 <= reg3;
    end
endmodule // sync_reset

```



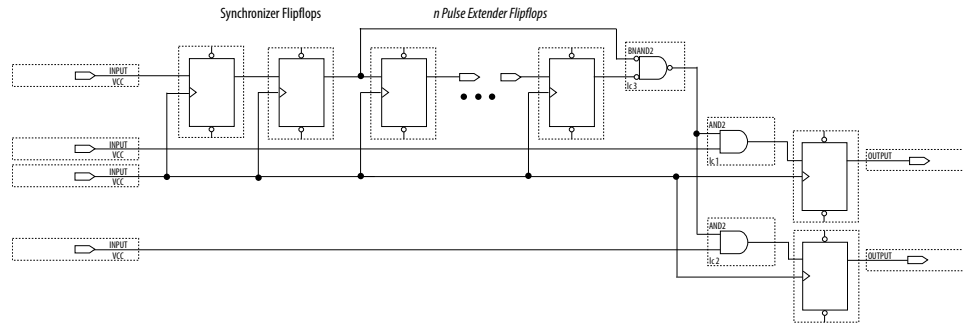
The SDC constraints are similar to the external synchronous reset, except that the input reset cannot be constrained because it is asynchronous. Cut the input path with a `set_false_path` statement to avoid these being considered as unconstrained paths.

#### Example 59. SDC Constraints for Internally Synchronized Reset

```
# Input clock - 100 MHz
create_clock [get_ports {clock}] \
    -name {clock} \
    -period 10.0 \
    -waveform {0.0 5.0}
# Input constraints on data
set_input_delay 7.0 \
    -max \
    -clock [get_clocks {clock}] \
    [get_ports {data_a data_b}]
set_input_delay 1.0 \
    -min \
    -clock [get_clocks {clock}] \
    [get_ports {data_a data_b}]
# Cut the asynchronous reset input
set_false_path \
    -from [get_ports {reset_n}] \
    -to [all_registers]
```

An issue with synchronous resets is their behavior with respect to short pulses (less than a period) on the asynchronous input to the synchronizer flipflops. This can be a disadvantage because the asynchronous reset requires a pulse width of at least one period wide to guarantee that it is captured by the first flipflop. However, this can also be viewed as an advantage in that this circuit increases noise immunity. Spurious pulses on the asynchronous input have a lower chance of being captured by the first flipflop, so the pulses do not trigger a synchronous reset. In some cases, you might want to increase the noise immunity further and reject any asynchronous input reset that is less than  $n$  periods wide to debounce an asynchronous input reset.

**Figure 47. Internally Synchronized Reset with Pulse Extender**



Junction dots indicate the number of stages. You can have more flipflops to get a wider pulse that spans more clock cycles.

Many designs have more than one clock signal. In these cases, use a separate reset synchronization circuit for each clock domain in the design. When you create synchronizers for PLL output clocks, these clock domains are not reset until you lock the PLL and the PLL output clocks are stable. If you use the reset to the PLL, this reset does not have to be synchronous with the input clock of the PLL. You can use an asynchronous reset for this. Using a reset to the PLL further delays the assertion of a synchronous reset to the PLL output clock domains when using internally synchronized resets.

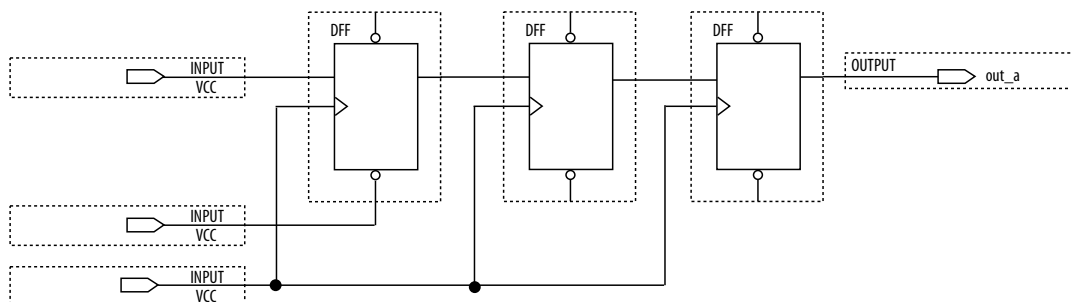
### 5.3.1.2 Using Asynchronous Resets

Asynchronous resets are the most common form of reset in circuit designs, as well as the easiest to implement. Typically, you can insert the asynchronous reset into the device, turn on the global buffer, and connect to the asynchronous reset pin of every register in the device.

This method is only advantageous under certain circumstances—you do not need to always reset the register. Unlike the synchronous reset, the asynchronous reset is not inserted in the datapath, and does not negatively impact the data arrival times between registers. Reset takes effect immediately, and as soon as the registers receive the reset pulse, the registers are reset. The asynchronous reset is not dependent on the clock.

However, when the reset is deasserted and does not pass the recovery ( $\mu t_{SU}$ ) or removal ( $\mu t_{H}$ ) time check (the Timing Analyzer recovery and removal analysis checks both times), the edge is said to have fallen into the metastability zone. Additional time is required to determine the correct state, and the delay can cause the setup time to fail to register downstream, leading to system failure. To avoid this, add a few follower registers after the register with the asynchronous reset and use the output of these registers in the design. Use the follower registers to synchronize the data to the clock to remove the metastability issues. You should place these registers close to each other in the device to keep the routing delays to a minimum, which decreases data arrival times and increases MTBF. Ensure that these follower registers themselves are not reset, but are initialized over a period of several clock cycles by “flushing out” their current or initial state.

**Figure 48. Asynchronous Reset with Follower Registers**



The following example shows the equivalent Verilog HDL code. The active edge of the reset is now in the sensitivity list for the procedural block, which infers a clock enable on the follower registers with the inverse of the reset signal tied to the clock enable. The follower registers should be in a separate procedural block as shown using non-blocking assignments.

**Example 60. Verilog HDL Code of Asynchronous Reset with Follower Registers**

```

module async_reset (
    input  clock,
    input  reset_n,
    input  data_a,
    output out_a,
);
    reg  reg1, reg2, reg3;
    assign out_a = reg3;
    always @ (posedge clock, negedge reset_n)
    begin
        if (!reset_n)
            reg1  <= 1'b0;
        else
            reg1  <= data_a;
    end
    always @ (posedge clock)
    begin
        reg2  <= reg1;
        reg3  <= reg2;
    end
endmodule // async_reset

```

You can easily constrain an asynchronous reset. By definition, asynchronous resets have a non-deterministic relationship to the clock domains of the registers they are resetting. Therefore, static timing analysis of these resets is not possible and you can use the `set_false_path` command to exclude the path from timing analysis. Because the relationship of the reset to the clock at the register is not known, you cannot run recovery and removal analysis in the Timing Analyzer for this path. Attempting to do so even without the false path statement results in no paths reported for recovery and removal.

**Example 61. SDC Constraints for Asynchronous Reset**

```

# Input clock - 100 MHz
create_clock [get_ports {clock}] \
    -name {clock} \
    -period 10.0 \
    -waveform {0.0 5.0}

```



```
# Input constraints on data
set_input_delay 7.0 \
  -max \
  -clock [get_clocks {clock}]\
  [get_ports {data_a}]
set_input_delay 1.0 \
  -min \
  -clock [get_clocks {clock}] \
  [get_ports {data_a}]
# Cut the asynchronous reset input
set_false_path \
  -from [get_ports {reset_n}] \
  -to [all_registers]
```

The asynchronous reset is susceptible to noise, and a noisy asynchronous reset can cause a spurious reset. You must ensure that the asynchronous reset is debounced and filtered. You can easily enter into a reset asynchronously, but releasing a reset asynchronously can lead to potential problems (also referred to as “reset removal”) with metastability, including the hazards of unwanted situations with synchronous circuits involving feedback.

### 5.3.1.3 Use Synchronized Asynchronous Reset

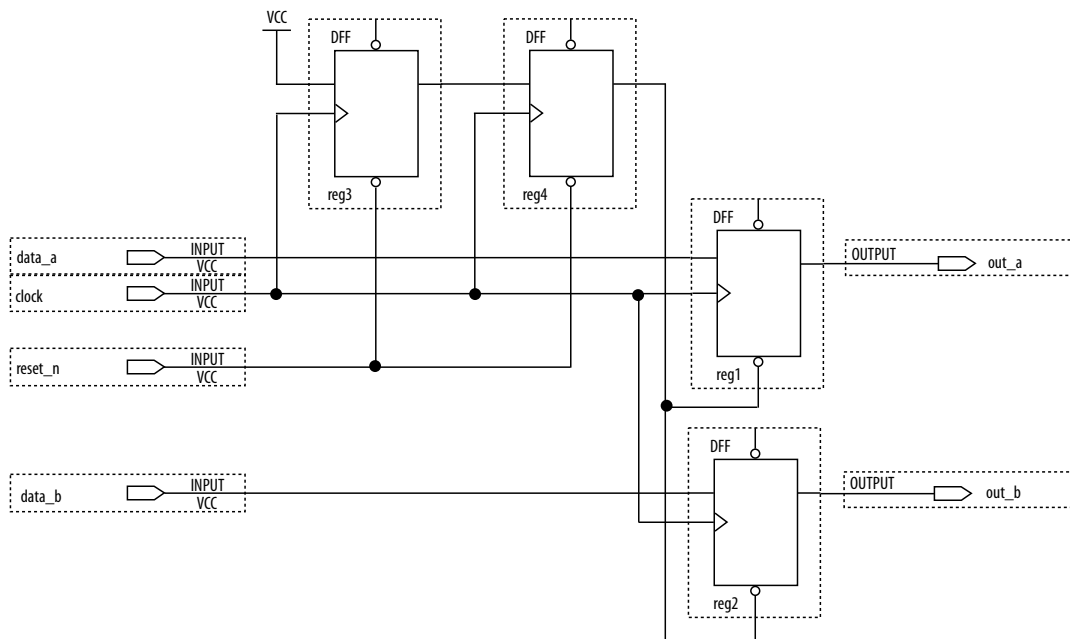
To avoid potential problems associated with purely synchronous resets and purely asynchronous resets, you can use synchronized asynchronous resets. Synchronized asynchronous resets combine the advantages of synchronous and asynchronous resets.

These resets are asynchronously asserted and synchronously deasserted. This takes effect almost instantaneously, and ensures that no datapath for speed is involved. Also, the circuit is synchronous for timing analysis and is resistant to noise.

The following example shows a method for implementing the synchronized asynchronous reset. You should use synchronizer registers in a similar manner as synchronous resets. However, the asynchronous reset input is gated directly to the CLRN pin of the synchronizer registers and immediately asserts the resulting reset. When the reset is deasserted, logic “1” is clocked through the synchronizers to synchronously deassert the resulting reset.



Figure 49. Schematic of Synchronized Asynchronous Reset



The following example shows the equivalent Verilog HDL code. Use the active edge of the reset in the sensitivity list for the blocks.

Example 62. Verilog HDL Code for Synchronized Asynchronous Reset

```

module sync_async_reset (
    input  clock,
    input  reset_n,
    input  data_a,
    input  data_b,
    output out_a,
    output out_b
);
    reg  reg1, reg2;
    reg  reg3, reg4;
    assign out_a  = reg1;
    assign out_b  = reg2;
    assign rst_n  = reg4;
    always @ (posedge clock, negedge reset_n)
    begin
        if (!reset_n)
            begin
                reg3    <= 1'b0;
                reg4    <= 1'b0;
            end
        else
            begin
                reg3    <= 1'b1;
                reg4    <= reg3;
            end
        end
    end
    always @ (posedge clock, negedge rst_n)
    begin
        if (!rst_n)
            begin
                reg1    <= 1'b0;
                reg2    <= 1'b0;
            end
    end
end

```

```
end
else
begin
    reg1    <= data_a;
    reg2    <= data_b;
end
end
endmodule // sync_async_reset
```

To minimize the metastability effect between the two synchronization registers, and to increase the MTBF, the registers should be located as close as possible in the device to minimize routing delay. If possible, locate the registers in the same logic array block (LAB). The input reset signal (`reset_n`) must be excluded with a `set_false_path` command:

```
set_false_path -from [get_ports {reset_n}] -to [all_registers]
```

The `set_false_path` command used with the specified constraint excludes unnecessary input timing reports that would otherwise result from specifying an input delay on the reset pin.

The instantaneous assertion of synchronized asynchronous resets is susceptible to noise and runt pulses. If possible, you should debounce the asynchronous reset and filter the reset before it enters the device. The circuit ensures that the synchronized asynchronous reset is at least one full clock period in length. To extend this time to  $n$  clock periods, you must increase the number of synchronizer registers to  $n + 1$ . You must connect the asynchronous input reset (`reset_n`) to the `CLRn` pin of all the synchronizer registers to maintain the asynchronous assertion of the synchronized asynchronous reset.

### 5.3.2 Use Global Clock Network Resources

Intel FPGAs provide device-wide global clock routing resources and dedicated inputs. Use the FPGA's low-skew, high fan-out dedicated routing where available.

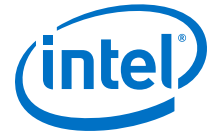
By assigning a clock input to one of these dedicated clock pins or with a Intel Quartus Prime assignment to assign global routing, you can take advantage of the dedicated routing available for clock signals.

In an ASIC design, you should balance the clock delay distributed across the device. Because Intel FPGAs provide device-wide global clock routing resources and dedicated inputs, there is no need to manually balance delays on the clock network.

You should limit the number of clocks in your design to the number of dedicated global clock resources available in your FPGA. Clocks feeding multiple locations that do not use global routing may exhibit clock skew across the device leading to timing problems. In addition, when you use combinational logic to generate an internal clock, it adds delays on the clock path. In some cases, delay on a clock line can result in a clock skew greater than the data path length between two registers. If the clock skew is greater than the data delay, you violate the timing parameters of the register (such as hold time requirements) and the design does not function correctly.

FPGAs offer a number of low-skew global routing resources to distribute high fan-out signals to help with the implementation of large designs with many clock domains. Many large FPGA devices provide dedicated global clock networks, regional clock networks, and dedicated fast regional clock networks. These clocks are organized into a hierarchical clock structure that allows many clocks in each device region with low





skew and delay. There are typically several dedicated clock pins to drive either global or regional clock networks, and both PLL outputs and internal clocks can drive various clock networks.

Intel Stratix 10 devices have a newer architecture. You can configure Intel Stratix 10 clocking resources to create efficiently balanced clock trees of various sizes, ranging from a single clock sector to the entire device. By default, the Intel Quartus Prime Software automatically determines the size and location of the clock tree. Alternatively, you can directly constrain the clock tree size and location either with a Clock Region assignment or by Logic Lock Regions.

To reduce clock skew in a given clock domain and ensure that hold times are met in that clock domain, assign each clock signal to one of the global high fan-out, low-skew clock networks in the FPGA device. The Intel Quartus Prime software automatically uses global routing for high fan-out control signals, PLL outputs, and signals feeding the global clock pins on the device. You can make explicit Global Signal logic option settings by turning on the **Global Signal** option setting. Use this option when it is necessary to force the software to use the global routing for particular signals.

*Note:* A Global Signal assignment only controls whether a signal should be promoted using the specified dedicated resources or not, but does not control which or how many resources get used.

To take full advantage of these routing resources, the sources of clock signals in a design (input clock pins or internally-generated clocks) need to drive only the clock input ports of registers. In older Intel device families, if a clock signal feeds the data ports of a register, the signal may not be able to use dedicated routing, which can lead to decreased performance and clock skew problems. In general, allowing clock signals to drive the data ports of registers is not considered synchronous design and can complicate timing closure.

### 5.3.3 Use Clock Region Assignments to Optimize Clock Constraints

The Intel Quartus Prime software determines how clock regions are assigned. You can override these assignments with Clock Region assignments to specify that a signal routed using global routing paths must use the specified clock region.

Use Clock Region assignments when you want to control the placement of the clock region for floorplanning reasons. For example, use a Clock Region assignment to ensure that a certain area of the device has access to a global signal, throughout your design iterations. A Clock Region assignment can also be used in cases of congestion involving global signal resources. By specifying a smaller clock region size, the assignment prevents a signal using spine clock resources in the excluded sectors that may be encountering clock-related congestion.

You can specify Clock Region assignments in the assignment editor.

#### Intel Arria 10 and Older Device Families

In device families with dedicated clock network resources and predefined clock regions, this assignment takes as its value the names of those Global, Regional, Periphery or Spine Clock regions. These region names are visible in Chip Planner by enabling the appropriate Clock Region layer in the **Layers Settings** dialog box. Examples of valid values include Regional Clock Region 1 or Periphery Clock Region 1.



When constraining a global signal to a smaller than normal region, for example, to avoid clock congestion, you may specify a clock region of a different type than the global resources being used. For example, a signal with a Global Signal assignment of `Global Clock`, but a Clock Region assignment of `Regional Clock Region 0`, constrains the clock to use global network routing resources, but only to the region covered by `Regional Clock Region 0`. To provide a finer level of control, you can also list multiple smaller clock regions, separated by commas. For example: `Periphery Clock Region 0, Periphery Clock Region 1` constrains a signal to only the area reachable by those two periphery clock networks.

### Intel Stratix 10 Devices

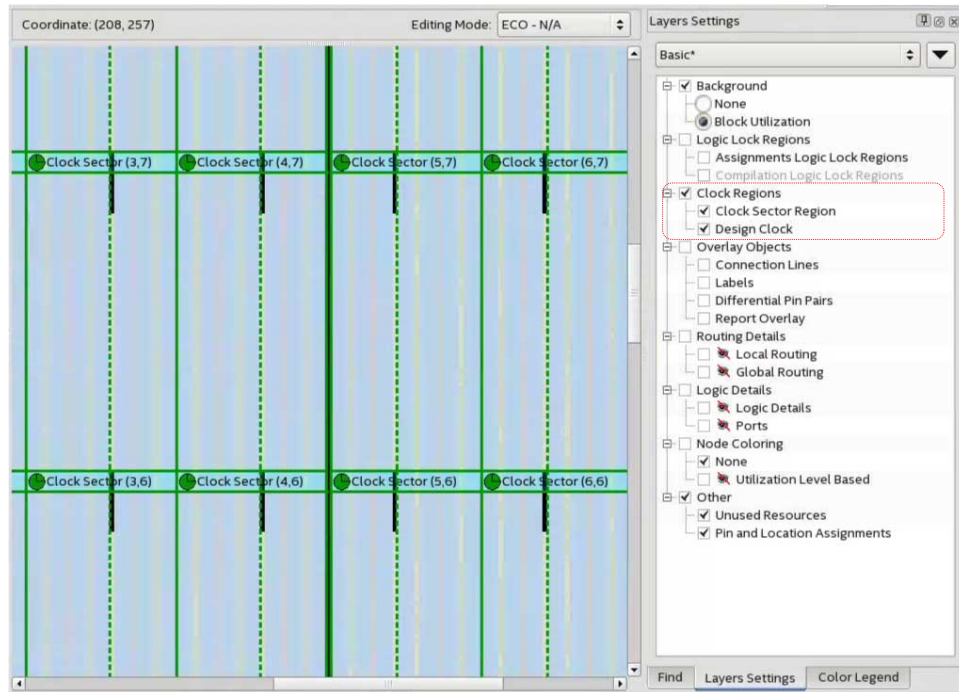
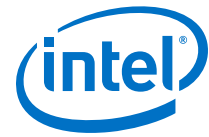
In Intel Stratix 10 devices, clock networks are constructed using programmable clock routing. As with other Intel device families, you can use Clock Region assignments for floorplanning, controlling the size and location of each clock tree.

Although the Intel Quartus Prime Pro Edition software generates balanced clock trees, there are sources of timing variation, such as process variation and jitter, which prevents clock trees from being perfectly skew balanced. Longer paths, with higher insertion delay, have more timing variation. However the Timing Analyzer can account for and eliminate some sources of variation in timing along common clock paths. In practice, this means that the size of the clock region has a significant impact on the worst-case skew of the clock tree; a larger clock tree experiences higher insertion delay and worst-case clock skew when compared to a smaller clock region. The distance between the clock region and the clock source also increases insertion delay, but the impact of distance on worst-case clock skew is much smaller than the impact of the size of the clock region.

One case to consider is when a design contains high-speed clock domains that are expected to grow during the design process. Specifying a clock region constraint to create a larger clock region than the compiler generates automatically helps ensure that timing closure is robust with higher clock insertion delays and clock skews.

An additional design consideration is the minimum pulse width constraint on clock signals. For a clock signal to propagate correctly on the Intel Stratix 10 clock network, a minimum delay must be met between the rising edge and falling edge of the clock pulse. If the Timing Analyzer cannot guarantee that this constraint is met, the clock signal may not propagate as expected under all operating conditions. This can happen when the delay variation on a clock path becomes too great. This situation does not normally occur, but may arise if clock signals are routed through core logic elements or core routing resources.

In designs that target Intel Stratix 10 devices, clock regions can be constrained to a rectangle whose dimensions are defined by the sector grid, as seen in the Clock Sector Region layer of the Chip Planner.



This assignment specifies the bottom left and top right coordinates of the rectangle in the format "SX# SY# SX# SY#". For example, "SX0 SY0 SX1 SY1" constrains the clock to a 2x2 region, from the bottom left of sector (0,0) to the top right of sector (1,1). For a constraint spanning only one sector, it is sufficient to specify the location of that sector, for example "SX1 SY1". The bounding rectangle can also be specified by the bottom left and top right corners in chip coordinates, for example, "X37 Y181 X273 Y324". However, such a constraint should be sector aligned (using sector coordinates guarantees this) or the Fitter automatically snaps to the smallest sector aligned rectangle that still encompasses the original assignment. The "SX# SY# SX# SY#"|"X# Y# X# Y#" strings are case-insensitive.

### 5.3.4 Avoid Asynchronous Register Control Signals

Avoid using an asynchronous load signal if the design target device architecture does not include registers with dedicated circuitry for asynchronous loads. Also, avoid using both asynchronous clear and preset if the architecture provides only one of these control signals.

Some Intel devices directly support an asynchronous clear function, but not a preset or load function. When the target device does not directly support the signals, the synthesis or placement and routing software must use combinational logic to implement the same functionality. In addition, if you use signals in a priority other than the inherent priority in the device architecture, combinational logic may be required to implement the necessary control signals. Combinational logic is less efficient and can cause glitches and other problems; it is best to avoid these implementations.

## 5.4 Implementing Embedded RAM

Intel’s dedicated memory architecture offers many advanced features that you can enable with Intel-provided IP cores. Use synchronous memory blocks for your design, so that the blocks can be mapped directly into the device dedicated memory blocks.

You can use single-port, dual-port, or three-port RAM with a single- or dual-clocking method. You should not infer the asynchronous memory logic as a memory block or place the asynchronous memory logic in the dedicated memory block, but implement the asynchronous memory logic in regular logic cells.

Intel memory blocks have different read-during-write behaviors, depending on the targeted device family, memory mode, and block type. Read-during-write behavior refers to read and write from the same memory address in the same clock cycle; for example, you read from the same address to which you write in the same clock cycle.

You should check how you specify the memory in your HDL code when you use read-during-write behavior. The HDL code that describes the read returns either the old data stored at the memory location, or the new data being written to the memory location.

In some cases, when the device architecture cannot implement the memory behavior described in your HDL code, the memory block is not mapped to the dedicated RAM blocks, or the memory block is implemented using extra logic in addition to the dedicated RAM block. Implement the read-during-write behavior using single-port RAM in Arria GX devices and the Cyclone and Stratix series of devices to avoid this extra logic implementation.

In many synthesis tools, you can specify that the read-during-write behavior is not important to your design; if, for example, you never read and write from the same address in the same clock cycle.

### Related Links

[Inferring RAM functions from HDL Code](#) on page 105

## 5.5 Document Revision History

**Table 32. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>Updated topic: Optimizing Timing Closure.</li> <li>Updated topic <i>Use Global Clock Network Resources</i> and added topic <i>Use Clock Region Assignments to Optimize Clock Constraints</i> for Intel Stratix 10support.</li> </ul>
2017.05.08	17.0.0	<ul style="list-style-type: none"> <li>Removed information about Integrated Synthesis.</li> <li>Removed information about quartus_drc.</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Implemented Intel rebranding.</li> </ul>
2016.05.03	16.0.0	<ul style="list-style-type: none"> <li>Replaced Internally Synchronized Reset code sample with corrected version.</li> <li>Removed information about deprecated physical synthesis options.</li> <li>Removed information about unsupported Design Assistant.</li> </ul>
<i>continued...</i>		



Date	Version	Changes
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li> </ul>
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
June 2014	14.0.0	Removed references to obsolete MegaWizard Plug-In Manager.
November 2013	13.1.0	Removed HardCopy device information.
May 2013	13.0.0	Removed PrimeTime support.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Added information to Reset Resources .
December 2010	10.1.0	<ul style="list-style-type: none"> <li>Title changed from Design Recommendations for Intel Devices and the Intel Quartus Prime Design Assistant.</li> <li>Updated to new template.</li> <li>Added references to Intel Quartus Prime Help for "Metastability" on page 9–13 and "Incremental Compilation" on page 9–13.</li> <li>Removed duplicated content and added references to Intel Quartus Prime Help for "Custom Rules" on page 9–15.</li> </ul>
July 2010	10.0.0	<ul style="list-style-type: none"> <li>Removed duplicated content and added references to Intel Quartus Prime Help for Design Assistant settings, Design Assistant rules, Enabling and Disabling Design Assistant Rules, and Viewing Design Assistant reports.</li> <li>Removed information from "Combinational Logic Structures" on page 5–4</li> <li>Changed heading from "Design Techniques to Save Power" to "Power Optimization" on page 5–12</li> <li>Added new "Metastability" section</li> <li>Added new "Incremental Compilation" section</li> <li>Added information to "Reset Resources" on page 5–23</li> <li>Removed "Referenced Documents" section</li> </ul>
November 2009	9.1.0	<ul style="list-style-type: none"> <li>Removed documentation of obsolete rules.</li> </ul>
March 2009	9.0.0	<ul style="list-style-type: none"> <li>No change to content.</li> </ul>
November 2008	8.1.0	<ul style="list-style-type: none"> <li>Changed to 8-1/2 x 11 page size</li> <li>Added new section "Custom Rules Coding Examples" on page 5–18</li> <li>Added paragraph to "Recommended Clock-Gating Methods" on page 5–11</li> <li>Added new section: "Design Techniques to Save Power" on page 5–12</li> </ul>
May 2008	8.0.0	<ul style="list-style-type: none"> <li>Updated Figure 5–9 on page 5–13; added custom rules file to the flow</li> <li>Added notes to Figure 5–9 on page 5–13</li> <li>Added new section: "Custom Rules Report" on page 5–34</li> <li>Added new section: "Custom Rules" on page 5–34</li> <li>Added new section: "Targeting Embedded RAM Architectural Features" on page 5–38</li> <li>Minor editorial updates throughout the chapter</li> <li>Added hyperlinks to referenced documents throughout the chapter</li> </ul>



**Related Links**

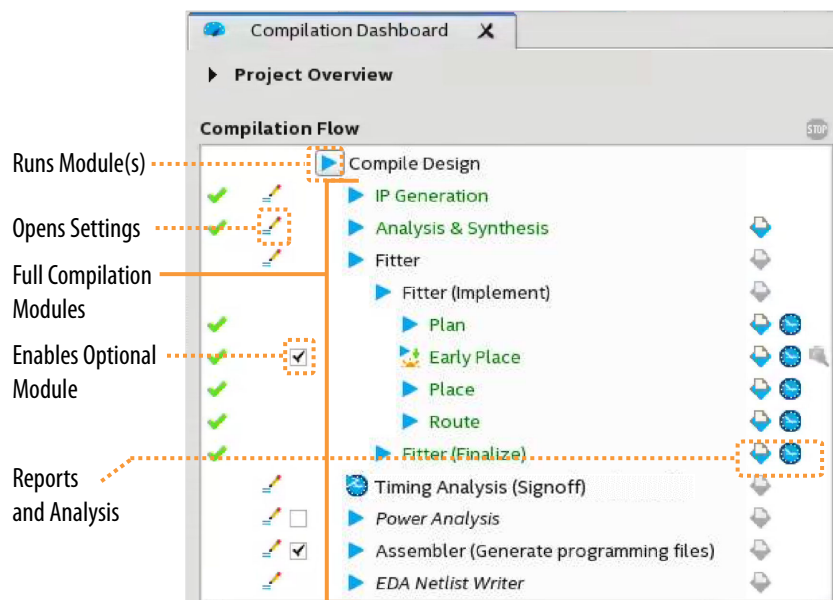
[Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

## 6 Design Compilation

The Intel Quartus Prime Compiler synthesizes, places, and routes your design before ultimately generating a device programming file. The Compiler supports a variety of high-level, HDL, and schematic design entry. The modules of the Compiler include IP Generation, Analysis & Synthesis, Fitter, Timing Analyzer, and Assembler.

**Figure 50. Intel Arria 10 Design in Compilation Dashboard**



The Intel Quartus Prime Pro Edition Compiler supports these advanced features:

- Latest compilation support for Intel Arria 10, Intel Cyclone 10 GX, and Intel Stratix 10 devices.
- Incremental Fitter optimization—analyze the design and optimize after Fitter each stage to maximize performance and shorten total compilation time.
- Hyper-Aware Design Flow—use Hyper-Retiming and Fast Forward compilation for the highest performance in Intel Stratix 10 devices.
- Partial Reconfiguration—supports dynamic reconfiguration of a portion of the FPGA, while the remaining FPGA continues to function.
- Block-Based Design Flows—enables preservation of design blocks within a project, and reuse of those design blocks in other projects.

## 6.1 Compilation Overview

The Compiler is modular, allowing you to run only the process that you need. Each Compiler module performs a specific function in the full compilation process. When you run any module, the Compiler runs any prerequisite modules automatically. The Compiler generates detailed reports and preserves a "snapshot" of the compilation results of each stage.

**Table 33. Compilation Modules**

Compilation Process	Description
IP Generation	Identifies the status and version IP components in the project.
Analysis & Synthesis	Synthesizes, optimizes, minimizes, and maps design logic to device resources. Analysis & Elaboration is a stage of Analysis & Synthesis. This stage checks for design file and project errors.
Fitter (Place & Route)	Assigns the placement and routing of the design to specific device resources, while honoring timing and placement constraints. The Fitter includes the following stages: <ul style="list-style-type: none"> <li>Plan—places all periphery elements (such as I/Os and PLLs) and determines a legal clock plan, without core placement or routing.</li> <li>Early Place—places all core elements in an approximate location to facilitate design planning. Finalizes clock planning for Intel Stratix 10 designs.</li> <li>Place—places all core elements in a legal location.</li> <li>Route—creates all routing between the elements in the design.</li> <li>Retime—moves (retimes) existing registers into Hyper-Registers for fine-grained performance improvement.<sup>(2)</sup></li> <li>Finalize—for Intel Arria 10 and Intel Cyclone 10 devices, converts unnecessary tiles to High-Speed or Low-Power. For Intel Stratix 10 devices, performs post-Route fix-up.</li> </ul>
Fast Forward Timing Closure Recommendations	Generates detailed reports that estimate performance gains achievable by making specific RTL modifications.
Timing Analyzer	Analyzes and validates the timing performance of all design logic.
Power Analysis	Optional module that estimates device power consumption. Specify the electrical standard on each I/O cell and the board trace model on each I/O standard in your design.
Assembler	Converts the Fitter's placement and routing assignments into a programming image for the FPGA device.
EDA Netlist Writer	Generates output files for use in other EDA tools.

### 6.1.1 Compilation Flows

The Intel Quartus Prime Pro Edition Compiler supports a variety of flows to help you maximize performance and minimize compilation processing time. The modular Compiler is flexible and efficient, allowing you to run all modules in sequence with a single command, or to run and optimize each stage of compilation separately.

As you develop and optimize your design, run only the Compiler stages that you need, rather than waiting for full compilation. Run full compilation only when your design is complete and you are ready to run all Compiler modules and generate a device programming image.

---

(2) Retiming and Fast Forward compilation available only for Intel Stratix 10 devices.





**Table 34. Compilation Flows**

Compiler Flow	Function
Early Place Flow	Places all core elements in an approximate location to facilitate design planning. Run Early Place to review initial high-level placement of design elements in the Chip Planner. This information is useful to guide your floorplanning decisions.
Implement Flow	Runs the Plan, Early Place, Place, Route, and Retime stages. Run this flow when you are ready to implement placement, routing, and retiming. <sup>(3)</sup>
Finalize Flow	Runs the Plan, Early Place, Place, Route, and Retime Compilation stages. Run this flow when you are ready to verify final timing closure results and generate a device programming file to implement the design in the target device.
Incremental Optimization Flow	Incremental optimization allows you to stop processing after each stage, analyze the results, and adjust settings or RTL before proceeding to the next compilation stage. This iterative flow optimizes at each stage, without waiting for full compilation results.
Hyper-Aware Design Flow	Combines automated register retiming (Hyper-Retiming), with implementation of targeted timing closure recommendations (Fast Forward Compilation), to maximize use of Hyper-Registers and drive the highest performance in Intel Stratix 10 devices.
Full Compilation Flow	Launches all Compiler modules in sequence to synthesize, fit, analyze final timing, and generate a device programming file.
Partial Reconfiguration	Reconfigures a portion of the FPGA dynamically, while the remaining FPGA design continues to function.
Block-Based Design Flows	Supports preservation and reuse of design blocks in one or more projects. You can reuse synthesized, placed, or routed design blocks within the same project, or export the block to other projects. Reusable design blocks can include device core or periphery resources.

**Related Links**

- [Incremental Optimization Flow](#) on page 194
- [Creating a Partial Reconfiguration Design](#)
- [Block-Based Design Flows](#)
- [Intel Stratix 10 High Performance Design Handbook](#)

**6.1.2 Design Synthesis**

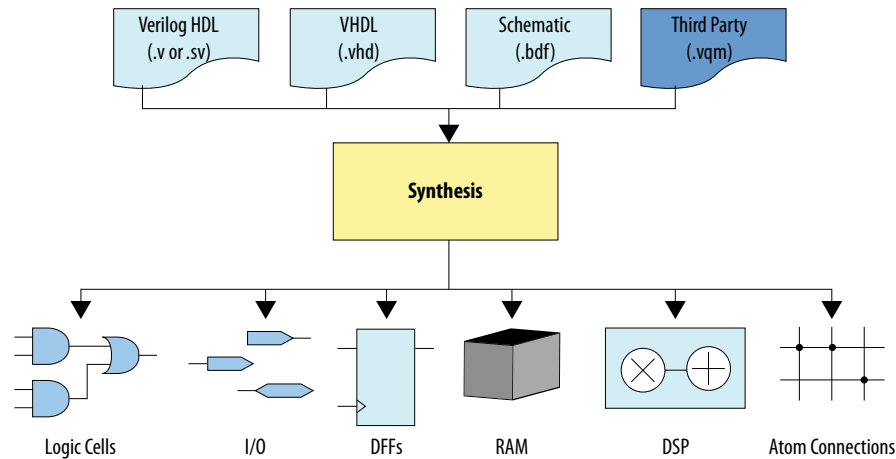
Design synthesis is the process that translates design source files into an atom netlist for mapping to device resources. The Intel Quartus Prime Compiler synthesizes standards-compliant Verilog HDL (.v), VHDL (.vhd), and SystemVerilog (.sv). The Compiler also synthesizes Block Design File (.bdf) schematic files, and the Verilog Quartus Mapping (.vqm) files generated by other EDA tools.

Synthesis examines the logical completeness and consistency of the design, and checks for boundary connectivity and syntax errors. Synthesis also minimizes and optimizes design logic. For example, synthesis infers D flip flops, latches, and state machines from "behavioral" languages, such as Verilog HDL, VHDL, and SystemVerilog. Synthesis may replace operators, such as + or -, with modules from the Intel Quartus Prime IP library, when advantageous. During synthesis, the Compiler may change or remove user logic and design nodes. Intel Quartus Prime synthesis minimizes gate count, removes redundant logic, and ensures efficient use of device resources.

---

<sup>(3)</sup> Retiming and Hyper-Aware design flow only for Intel Stratix 10 devices.

Figure 51. Design Synthesis



At the end of synthesis, the Compiler generates an atom netlist. Atom refers to the most basic hardware resource in the FPGA device. Atoms include logic cells organized into look-up tables, D flip flops, I/O pins, block memory resources, DSP blocks, and the connections between the atoms. The atom netlist is a database of the atom elements that design synthesis requires to implement the design in silicon.

The Analysis & Synthesis module of the Compiler synthesizes design files and creates one or more project databases for each design partition. You can specify various settings that affect synthesis processing.

The Compiler preserves the results of Analysis & Synthesis in the synthesis snapshot.

### 6.1.3 Design Place and Route

The Compiler's Fitter module (`quartus_fit`) performs design placement and routing. During place and route, the Fitter determines the best placement and routing of logic in the target FPGA device, while respecting any Fitter settings or constraints that you specify.

By default, the Fitter selects appropriate resources, interconnection paths, and pin locations. If you assign logic to specific device resources, the Fitter attempts to match those requirements, and then fits and optimizes any remaining unconstrained design logic. If the Fitter cannot fit the design in the current target device, the Fitter terminates compilation and issues an error message.

The Intel Quartus Prime Pro Edition Fitter introduces a hybrid placement technique that combines analytical and annealing placement techniques. Analytical placement determines an initial mathematical starting placement. The annealing technique then fine-tunes logic block placement in high resource utilization scenarios.

The Intel Quartus Prime Pro Edition Compiler allows control and optimization of each individual Fitter stage, including the Plan, Early Place, Place, and Route stages. After running a Fitter stage, view detailed report data and analyze the timing of that stage. The Compiler preserves the results of Fitter stages in the planned, early placed, placed, routed, retimed, and final snapshots.



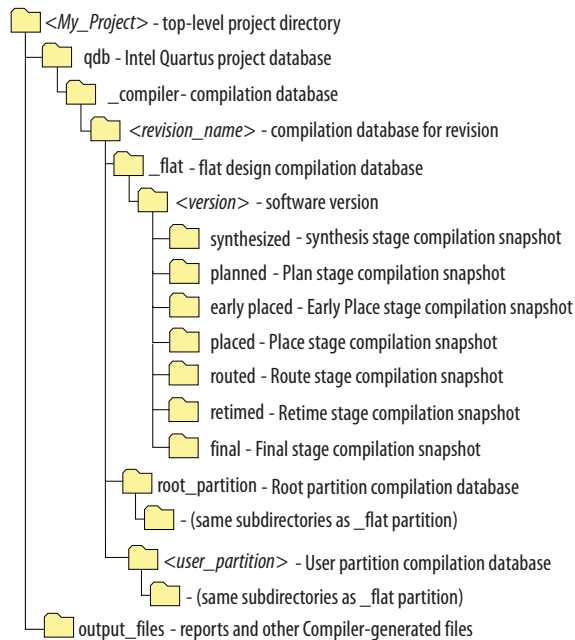
### Related Links

- [Running the Fitter](#) on page 192
- [Viewing Fitter Reports](#) on page 204

## 6.1.4 Compilation Hierarchy

The Intel Quartus Prime Pro Edition Compiler generates a hierarchical project structure that isolates results of each compilation stage, for each design entity. For example, the `synthesized` directory contains a snapshot of the Analysis & Synthesis stage. If you use design partitions, such as in block-based design, the Compiler also isolates the results for each design partition. The Compiler fully preserves routing and placement within a partition. Changes to other portions of the design hierarchy do not impact the partition. This hierarchical structure allows you to optimize specific design elements without impacting placement and routing in other partitions. The hierarchical project structure also supports distributed work groups and compilation processing across multiple machines.

**Figure 52. Hierarchical Project Structure (Intel Stratix 10 Design)**



### Related Links

#### [Block-Based Design Flows](#)

for information on team-based design and design block reuse

## 6.1.5 Reducing Compilation Time

The Intel Quartus Prime Pro Edition software supports various strategies to reduce overall design compilation time. Running a full compilation including all Compiler modules on a large design can be time consuming. Use any the following techniques to reduce the overall compilation times of your design:

- Rapid Recompile of changed blocks—the Compiler reuses previous compilation results and does not reprocess unchanged design blocks.  
*Note:* Rapid Recompile does not support Intel Stratix 10 devices.
- Parallel compilation—the Compiler detects and uses multiple processors to reduce compilation time (for systems with multiple processor cores).
- Incremental optimization—breaks compilation into separate stages, allowing iterative analysis of results and optimization of settings at various compilation stages, prior to running a full compilation.

### 6.1.6 Programming File Generation

The Compiler's Assembler module generates files for device programming. Run the Assembler automatically as part of a full compilation, or run the Assembler module independently after design place and route. After running the Assembler, use the Programmer to download configuration data to a device. The Assembler generates one or more of the following files according to your specification in the **Device & Pin Options** dialog box.

**Table 35. Assembler Generated Programming Files**

Programming File	Description
SRAM Object Files (.sof)	A binary file containing the data for configuring all SRAM-based Intel FPGA devices.
Programmer Object Files (.pof)	A binary file containing the data for programming an EEPROM-based Intel configuration device. For example, the EPCS16 and EPCS64 devices, which configure SRAM-based Intel FPGA devices.
Hexadecimal (Intel-Format) Output Files (.hexout)	Contains configuration data that you can program into a parallel data source, such as an EPROM or a mass storage device, which configures an SRAM-based Intel FPGA device.
Raw Binary Files (.rbf)	Contains configuration data that an intelligent external controller uses to configure an SRAM-based Intel FPGA device.
Tabular Text Files (.ttf)	Contains configuration data that an intelligent external controller uses to configure an SRAM-based Intel FPGA device.
Serial Vector Format File (.svf)	<i>Note:</i> Generation of these files not available for Intel Stratix 10 designs.

#### Related Links

[Generating Programming Files](#) on page 222

## 6.2 Running Full Compilation

Use these steps to run a full compilation of an Intel Quartus Prime project. A full compilation includes IP Generation, Analysis & Synthesis, Fitter, Timing Analyzer, and any optional modules you enable.

1. Before running a full compilation, specify any of the following project settings:



- To specify the target FPGA device or development kit, click **Assignments > Device**.
  - To specify device and pin options for the target FPGA device, click **Assignments > Device > Device and Pin Options**.
  - To specify options that affect compilation processing time and netlist preservation, click **Assignments > Settings > Compilation Process Settings**.
  - To specify synthesis algorithm and other **Advanced Settings** for synthesis and fitting, click **Assignments > Settings > Compiler Settings**.
  - To specify required timing conditions for proper operation of your design, click **Tools > Timing Analyzer**.
2. To run full compilation, click **Processing > Start Compilation**.
- Note:* Early Place does not run during full compilation by default. To enable Early Place during full compilation, click **Assignments > Settings > Compiler Settings > Advanced Settings (Fitter) > Run Early Place during compilation** option.

#### Related Links

- [Interface Planning](#)
- [The Timing Analyzer](#)
- [Managing Device I/O Pins](#)

## 6.3 Running Synthesis

Run design synthesis as part of a full compilation, or as an independent process. Before running synthesis, specify settings that control synthesis processing. The Messages window dynamically displays processing information, warnings, or errors. Following Analysis and Synthesis processing, the Synthesis report provides detailed information about the synthesis of each design partition.

To run synthesis:

1. Create or open an Intel Quartus Prime project with valid design files for compilation.
2. Before running synthesis, specify any of the following settings and constraints that impact synthesis:
  - To specify options for the synthesis of Verilog HDL input files, click **Assignments > Settings > Verilog HDL Input**.
  - To specify options for the synthesis of VHDL input files, click **Assignments > Settings > VHDL Input**.
  - To specify options that affect compilation processing time, click **Assignments > Settings > Compilation Process Settings**.
  - To specify advanced synthesis settings, click **Assignments > Settings > Compiler Settings**, and then click **Advanced Settings (Synthesis)**. Optionally, enable **Timing-Driven Synthesis** to account for timing constraints during synthesis.
3. To run synthesis, click **Synthesis** on the Compilation Dashboard.

### Related Links

[Synthesis Settings Reference](#) on page 229

## 6.3.1 Preserve Registers During Synthesis

Intel Quartus Prime synthesis minimizes gate count, merges redundant logic, and ensures efficient use of device resources. If you need to preserve specific registers through synthesis processing, you can specify any of the following entity-level assignments. Use **Preserve Registers in Synthesis** or **Preserve Fan-Out Free Register Node** to allow Fitter optimization of the preserved registers. **Preserve Registers** restricts Fitter optimization of the preserved registers. Specify synthesis preservation assignments by clicking **Assignments** > **Assignment Editor**, by modifying the .qsf file, or by specifying synthesis attributes in your RTL.

**Table 36. Synthesis Preserve Options**

Assignment	Description	Allows Fitter Optimization?	Assignment Syntax
<b>Preserve Registers in Synthesis</b>	Prevents removal of registers during synthesis. This settings does not affect retiming or other optimizations in the Fitter.	Yes	<ul style="list-style-type: none"> <li>PRESERVE_REGISTER_SYN_ONLY ON Off -to &lt;entity&gt; (.qsf)</li> <li>preserve_syn_only or syn_preservesyn_only (synthesis attributes)</li> </ul>
<b>Preserve Fan-Out Free Register Node</b>	Prevents removal of assigned registers without fan-out during synthesis.	Yes	<ul style="list-style-type: none"> <li>PRESERVE_REGISTER_FANOUT_FREE_NODE ON Off -to &lt;entity&gt; (.qsf)</li> <li>no_prune on (synthesis attribute)</li> </ul>
<b>Preserve Registers</b>	Prevents removal and sequential optimization of assigned registers during synthesis. Sequential netlist optimizations can eliminate redundant registers and registers with constant drivers.	No	<ul style="list-style-type: none"> <li>PRESERVE_REGISTER ON Off -to &lt;entity&gt; (.qsf)</li> <li>preserve, syn_preserve, or keep on (synthesis attributes)</li> </ul>

## 6.3.2 Enabling Timing-Driven Synthesis

Timing-driven synthesis directs the Compiler to account for your timing constraints during synthesis. Timing-driven synthesis runs initial timing analysis to obtain netlist timing information. Synthesis then focuses performance efforts on timing-critical design elements, while optimizing non-timing-critical portions for area.

Timing-driven synthesis preserves timing constraints, and does not perform optimizations that conflict with timing constraints. Timing-driven synthesis may increase the number of required device resources. Specifically, the number of adaptive look-up tables (ALUTs) and registers may increase. The overall area can increase or decrease. Runtime and peak memory use increases slightly.

Intel Quartus Prime Pro Edition runs timing-driven synthesis by default. To enable or disable this option manually, click **Assignments** > **Settings** > **Compiler Settings** > **Advanced Settings (Synthesis)**.

### Related Links

- [Running Synthesis](#) on page 189
- [Synthesis Language Support](#) on page 223



### 6.3.3 Enabling Multi-Processor Compilation

The Compiler can detect and use multiple processors to reduce total compilation time. You specify the number of processors the Compiler uses. The Intel Quartus Prime software can use up to 16 processors to run algorithms in parallel. The Compiler uses parallel compilation by default. To reserve some processors for other tasks, specify a maximum number of processors that the software uses.

This technique reduces the compilation time by up to 10% on systems with two processing cores, and by up to 20% on systems with four cores. When running timing analysis independently, two processors reduce the timing analysis time by an average of 10%. This reduction reaches an average of 15% when using four processors.

The Intel Quartus Prime software does not necessarily use all the processors that you specify during a given compilation. Additionally, the software never uses more than the specified number of processors. This fact enables you to work on other tasks without slowing down your computer. The use of multiple processors does not affect the quality of the fit. For a given Fitter seed, and given **Maximum processors allowed** setting on a specific design, the fit is exactly the same and deterministic. This remains true, regardless of the target machine, and the number of available processors. Different **Maximum processors allowed** specifications produce different results of the same quality. The impact is similar to changing the Fitter seed setting.

To enable multiprocessor compilation, follow these steps:

1. Open or create an Intel Quartus Prime project.
2. To enable multiprocessor compilation, click **Assignments > Settings > Compilation Process Settings**.
3. Under **Parallel compilation**, specify options for the number of processors the Compiler uses.
4. View detailed information about processor in the Parallel Compilation report following compilation.

To specify the number of processors for compilation at the command line, use the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS <value>
```

In this case, *<value>* is an integer from 1 to 16.

If you want the Intel Quartus Prime software to detect the number of processors and use all the processors for the compilation, include the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
```

*Note:* The Compiler detects Intel Hyper-Threading as a single processor. If your system includes a single processor with Intel Hyper-Threading, set the number of processors to one. Do not use the Intel Hyper-Threading feature for Intel Quartus Prime compilations.

### 6.3.4 Synthesis Reports

The Compilation Report window opens automatically during compilation processing. The Report window displays detailed synthesis results for each partition in the current project revision.

Figure 53. Synthesis Reports

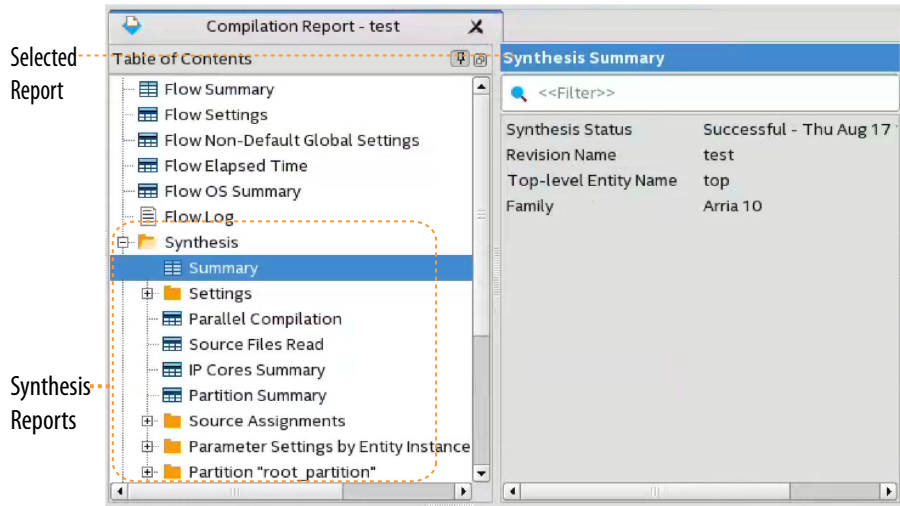


Table 37. Synthesis Reports (Design Dependent)

Generated Report	Description
Summary	Shows summary information about synthesis, such as the status, date, software version, entity name, device family, timing model status, and various types of logic utilization.
Synthesis Settings	Lists the values of all synthesis settings during design processing.
Parallel Compilation	Lists specifications for any use of parallel processing during synthesis.
Resource Utilization By Entity	Lists the quantity of all types of logic usage for each entity in design synthesis.
Multiplexer Restructuring Statistics	Provides statistics for the amount of multiplexer restructuring that synthesis performs.
IP Cores Summary	Lists details about each IP core instance in design synthesis. Details include IP core name, vendor, version, license type, entity instance, and IP include file.
Synthesis Source Files Read	Lists details about all source files in design synthesis. Details include file path, file type, and any library information.
Resource Usage Summary for Partition	Lists the quantity of all types of logic usage for each design partition in design synthesis.
RAM Summary for Partition	Lists RAM usage details for each design partition in design synthesis. Details include the name, type, mode, and density.
Register Statistics	Lists the number of registers using various types of global signals.
Synthesis Messages	Lists all information, warning, and error messages that report conditions observed during the Analysis & Synthesis process.

## 6.4 Running the Fitter

The Compiler's Fitter module performs all stages of design place and route, including the Plan, Early Place, Place, Route and Retime stages. Run all stages of the Fitter as part of a full design compilation, or run any Fitter stage independently after design synthesis. You can analyze the results of some Fitter stages, while downstream stages are still running. Before running the Fitter, you specify settings that tailor placement and routing results for your requirements.





1. Specify initial Fitter constraints:
  - a. To assign device I/O pins, click **Assignments > Pin Planner**.
  - b. To assign device periphery, clocks, and I/O interfaces, click **Tools > Interface Planner**.
  - c. To constrain logic placement regions, click **Tools > Chip Planner**.
  - d. To specify general performance, power, or logic usage focus for fitting, click **Assignments > Settings > Compiler Settings**.
  - e. To fine-tune place and route with advanced Fitter options, click **Assignments > Settings > Compiler Settings > Advanced Settings (Fitter)**.
2. To run one or more stages of the Fitter, click any of the following commands on the Compilation Dashboard:
  - To begin device periphery placement and routing, click **Plan**.
  - To run early placement, click **Early Place**.
  - To fully place design logic, click **Place**.
  - To fully route the design, click **Route**.
  - To retime ALM registers into Hyper-Registers, click **Retime**.<sup>(4)</sup>
  - To run the Implement flow (Plan, Place, Route, and Retime stages), click **Fitter (Implement)**.
  - To run the Finalize flow (Plan, Early Place, Place, Route, Retime, and Finalize stages), click **Fitter (Finalize)**.
  - To run all Fitter stages in sequence, click **Fitter**.

**Related Links**

- [Fitter Settings Reference](#) on page 236
- [Step 2: Review Retiming Results](#) on page 211

### 6.4.1 Fitter Stage Commands

Launch Fitter processes from the Processing menu or Compilation Dashboard.

**Table 38. Fitter Stage Commands**

Command	Description
<b>Fitter (Implement)</b>	Runs the Plan, Early Place, Place, Route, and Retime stages.
<b>Start Fitter (Plan)</b>	Loads synthesized periphery placement data and constraints, and assigns periphery elements to device I/O resources. After this stage, you can run post-Plan timing analysis to verify timing constraints, and validate cross-clock timing windows. View the placement and properties of periphery (I/O) and perform clock planning for Intel Arria 10 and Intel Cyclone 10 designs. This command creates the planned snapshot.
<b>Start Fitter (Early Place)</b>	Places all core elements in an approximate location to facilitate design planning. After this stage, the Chip Planner displays initial high-level placement of design elements. The Compilation reports identifies high fan-out signals that increase placement complexity. Use this information to guide your floorplanning decisions. For Intel Stratix 10 designs, you can also do early clock planning after this stage.

*continued...*

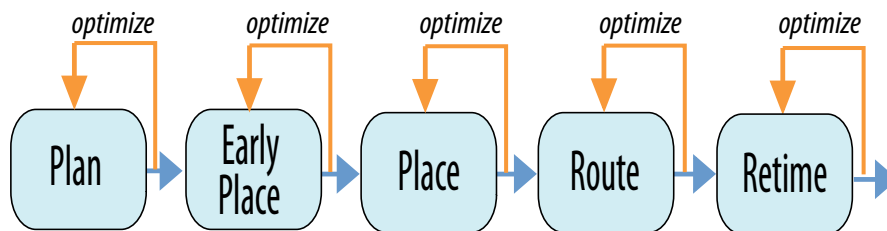
<sup>(4)</sup> Retime available for Intel Stratix 10 devices only.

Command	Description
	This command creates the early placed snapshot. Early Place does not run during the full compilation flow by default, but you can enable by default or run directly from the Compilation Dashboard.
<b>Start Fitter (Place)</b>	Places all core elements in a legal location. This command creates the placed snapshot.
<b>Start Fitter (Route)</b>	Creates all routing between the elements in the design. After this stage, validate delay chain settings and analyze routing resources. Perform detailed setup and hold timing closure in the Timing Analyzer and view routing congestion via the Chip Planner. This command creates the routed snapshot.
<b>Start Fitter (Retime)</b>	Retimes existing registers in the design into Hyper-Registers to increase performance by removing retiming restrictions and eliminate critical paths. The Compiler may report hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths. This command creates the retimed snapshot.
<b>Start Fitter (Finalize)</b>	Performs post-routing optimization on the design. This stage converts unneeded tiles from High Speed to Low Power. This command creates the final snapshot. For Intel Stratix 10 designs, the Fitter also runs post-route fix-up to correct any short path hold violations remaining from retiming.

### 6.4.2 Incremental Optimization Flow

Intel Quartus Prime Pro Edition supports incremental optimization at each stage of design compilation. In incremental optimization, you run and optimize each compilation stage independently before running the next compilation module in sequence. The Compiler preserves the results of each stage as a snapshot for analysis. When you make changes to your design or constraints, the Compiler only runs stages impacted by the change. Following synthesis or any Fitter stage, view results and perform timing analysis. Modify design RTL or Compiler settings, as needed. Then, re-run synthesis or the Fitter and evaluate the results of these changes. Repeat this process until the module performance meets requirements. This flow maximizes the results at each stage, without waiting for full compilation results.

**Figure 54. Incremental Optimization Flow**



**Table 39. Incremental Optimization at Fitter Stages**

Fitter Stage	Incremental Optimization
Plan	After this stage, you can run post-Plan timing analysis to verify timing constraints, and validate cross-clock timing windows. View the placement and properties of periphery (I/O) and perform clock planning for Intel Arria 10 and Intel Cyclone 10 designs.
Early Place	After this stage, the Chip Planner can display initial high-level placement of design elements. Use this information to guide your floorplanning decisions. For Intel Stratix 10 designs, you can also do early clock planning after running this stage.
<i>continued...</i>	



Fitter Stage	Incremental Optimization
Place	After this stage, validate resource and logic utilization in the Compilation Reports, and review placement of design elements in the Chip Planner.
Route	After this stage, perform detailed setup and hold timing closure in the Timing Analyzer, and view routing congestion via the Chip Planner.
Retime	After this stage, review the Retiming results in the Fitter report and correct any restrictions limiting further retiming optimization. <sup>(5)</sup>

### 6.4.2.1 Early Place Flow

Early Place begins assigning core logic to device resources. Run **Early Place** to quickly view the effect of iterative floorplanning changes, without waiting for full placement or full compilation. The Compiler preserves a snapshot of the Early Place results.

Following Early Place, click the Timing Analyzer icon to validate your `.sdc` constraints. Do not use the Early Place timing results to compare with Final timing results, as timing between early snapshots and the final snapshot are not well correlated. Early Place runs automatically during Fitter processing if you enable the **Early Place** stage on the compilation dashboard, or by enabling **Settings > Compiler Settings > Fitter Settings (Advanced) > Run Early Place During Compilation**.

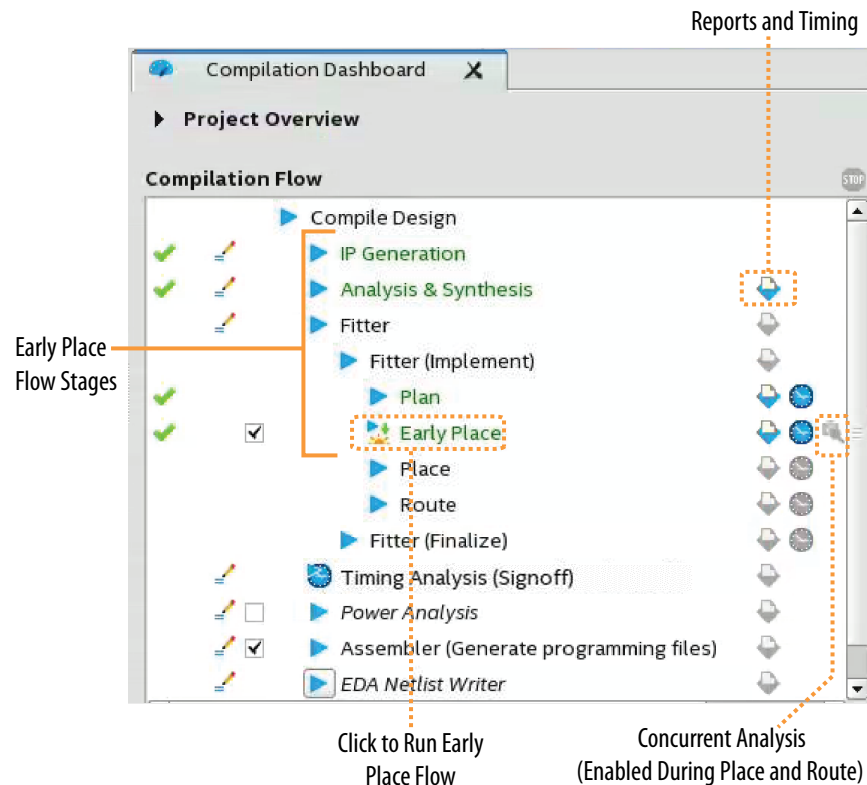
#### Concurrent Analysis of Early Place Results

If you run the Fitter (or **Place** or **Route** stages) without previously running **Early Place**, you can access the Early Place results while downstream Fitter stages are still running. Click the **Concurrent Analysis** icon on the Dashboard to analyze Early Place timing while the Fitter continues processing. You cannot modify timing constraints during concurrent analysis. However, stop compilation processing at any time, modify your `.sdc` constraints, and then click the **Timing Analyzer** icon to analyze the design with the modified constraints.

---

<sup>(5)</sup> Retiming available only for Intel Stratix 10 devices.

Figure 55. Early Place Flow in Compilation Dashboard



### 6.4.2.2 Running late\_place After Early Place

After running the Early Place stage, you can run `late_place`, rather than the full Place stage, to reduce total compilation time. `late_place` skips the placements that Early Place makes. The Place stage includes the Early Place and `late_place` stages. There is no GUI support for the Fitter's `late_place` option. The `late_place` option is only available at the command line, after running the Early Place stage from the GUI or command-line. Running `late_place` generates the placed snapshot. Access command-line help to display details about the `late_place` argument.

**Note:** Type `quartus_fit -help=late_place` for command-line help on this argument.

To run `late_place` after Early Place:

1. To run the Early Place stage and generate the Early Place snapshot, perform one of the following:
  - To run Early Place in the GUI, click **Early Place** on the Compilation Dashboard. The Compiler runs any required prerequisite stages.
  - To run Early Place (and prerequisite stages) at the command-line, run the following commands. The "-" character equals double hyphens:

```
quartus_ipgenerate <design_name>
quartus_syn <design name>
quartus_fit -plan <design name>
quartus_fit -early_place <design name>
```



*Note:* You must generate the early placed snapshot before running `late_place`, or the Fitter reports an error.

2. View Early Place results in the Early Placed Fitter reports of the Compilation Report, and in the Chip Planner (**Tools** > **Chip Planner**).
3. When satisfied with the Early Place results, type one of the following commands to continue to the `late_place` stage and beyond. View `late_place` results and processing messages in the `<design name>.fit.place.rpt` file.

- ```
quartus_fit -late_place <design name>  
(runs late_place)
```
- ```
quartus_fit -late_place -route <design name>  
(runs late_place and route)
```
- ```
quartus_fit -late_place -route -finalize <design name>  
(runs late_place and finalize)
```

### 6.4.3 Analyzing Fitter Snapshots

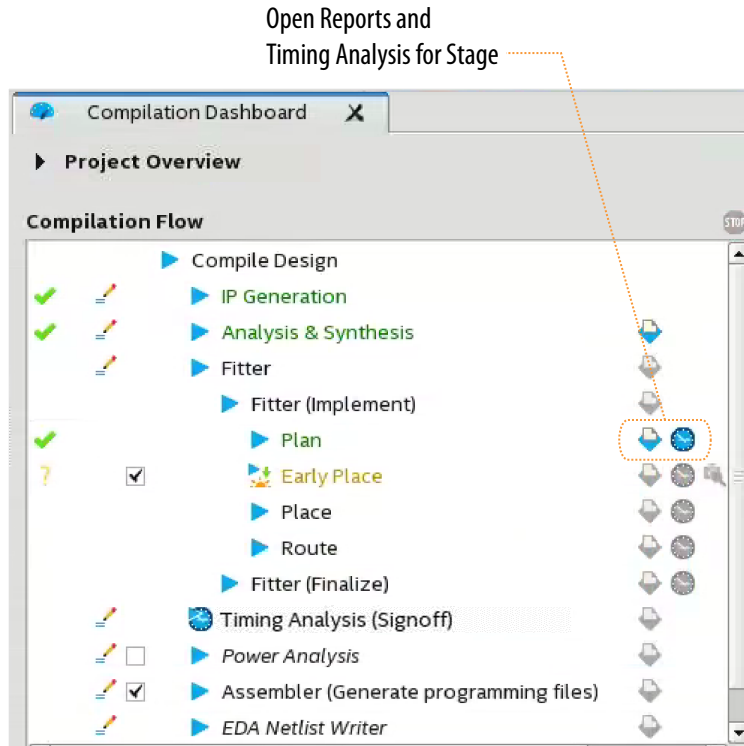
Analyze the results of Fitter stages to evaluate your design before running the next stage, or before running a full compilation. Use this technique to isolate potential problems and reduce the overall time you spend running design compilation. The following topics describe typical use cases for analyzing Fitter snapshots.

#### 6.4.3.1 Validating SDC Constraints after the Plan Stage

The Fitter's Plan stage performs initial validation of your project's `.sdc` constraints. The Compiler generates messages during the plan stage that warn you about any possible invalid `.sdc` constraints. Stop compilation following the Plan stage to validate and make any necessary changes to `.sdc` constraints, before moving on to the next Fitter stage. To validate `.sdc` constraints after the Plan stage, follow these steps:

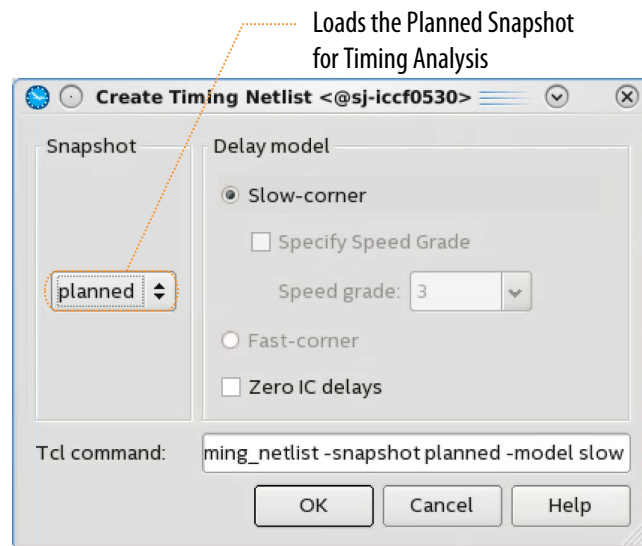
1. To run the Fitter's Plan stage, click **Plan** on the Compilation Dashboard. The Compiler automatically runs prerequisite compilation stages, if necessary.
2. On the Compilation Dashboard, click the **Timing Analyzer** icon adjacent to the Fitter stage. The **Create Timing Netlist** dialog box appears and loads the corresponding stage snapshot.

Figure 56. Plan Stage Timing Analyzer Icon in Compilation Dashboard



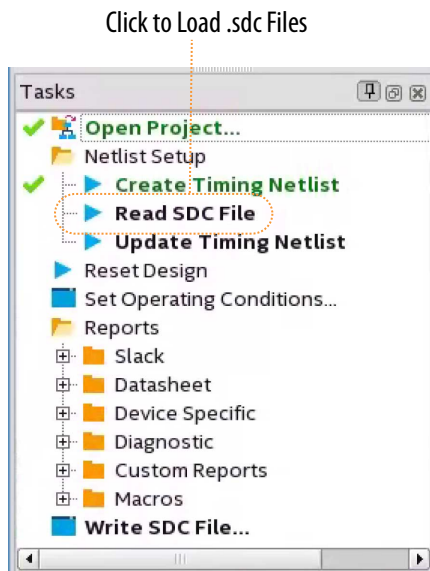
3. In the **Create Timing Netlist** dialog box, click **OK**. The planned database loads in the Timing Analyzer.

Figure 57. Planned Snapshot in Create Timing Netlist Dialog Box



4. On the **Tasks** pane, click **Read SDC File**. The Timing Analyzer reads and processes any `.sdc` files. For multiple `.sdc` files, the report also includes the `.sdc` processing sequence.

Figure 58. Read SDC File Command



5. To report the .sdc constraints that apply to the project, click **Report SDC** under the **Diagnostic** folder, in the **Tasks** pane.

Figure 59. SDC File List Report

| SDC File List |                |          |        |                          |                 |
|---------------|----------------|----------|--------|--------------------------|-----------------|
|               | SDC File Path  | Instance | Status | Read at                  | Processing Time |
| 1             | clocks.sdc     |          | OK     | Tue Aug 15 15:41:56 2017 | 00:00:00        |
| 2             | io.sdc         |          | OK     | Tue Aug 15 15:41:56 2017 | 00:00:00        |
| 3             | exceptions.sdc |          | OK     | Tue Aug 15 15:41:56 2017 | 00:00:00        |

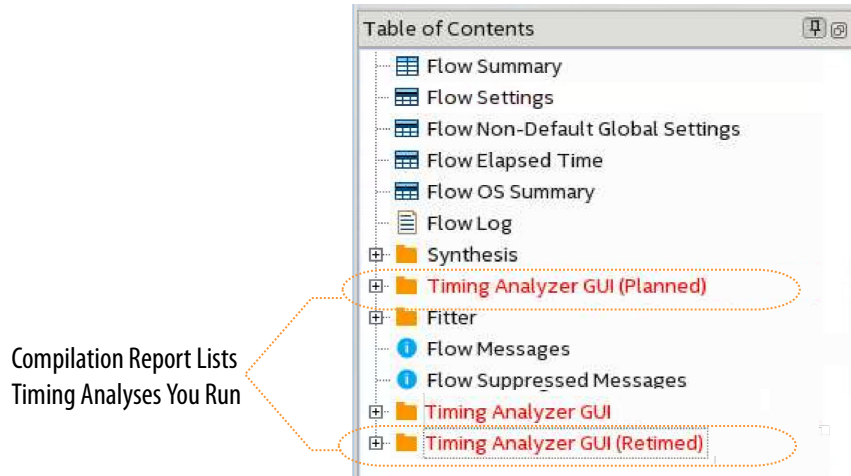
6. Conversely, to report the constraints in the .sdc files that the Timing Analyzer ignores, click **Report Ignored Constraints** under the **Diagnostic** folder, in the **Tasks** pane.
7. To report all paths in your design that have no constraints, click **Report Unconstrained Paths** under the **Diagnostic** folder, in the **Tasks** pane.

Figure 60. Unconstrained Paths Summary

| Unconstrained Paths Summary |                                 |       |      |  |
|-----------------------------|---------------------------------|-------|------|--|
|                             | Property                        | Setup | Hold |  |
| 1                           | Illegal Clocks                  | 0     | 0    |  |
| 2                           | Unconstrained Clocks            | 0     | 0    |  |
| 3                           | Unconstrained Input Ports       | 1     | 1    |  |
| 4                           | Unconstrained Input Port Paths  | 6     | 6    |  |
| 5                           | Unconstrained Output Ports      | 0     | 0    |  |
| 6                           | Unconstrained Output Port Paths | 0     | 0    |  |

The Compilation Report displays the Timing Analysis that you run for each stage.

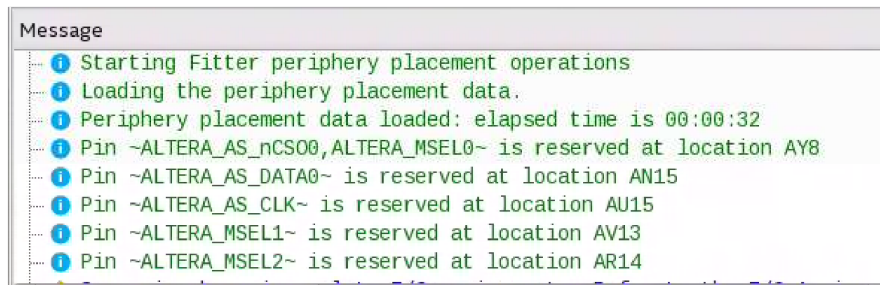
Figure 61. Plan and Retime Stage Timing Analysis Reports in Compilation Report



### 6.4.3.2 Validating Periphery (I/O) after the Plan Stage

The Compiler begins periphery placement during the plan stage, and reports data about periphery elements, such as I/O pins and PLLs. After the Plan stage, view the Compilation Report to evaluate the placement of periphery elements before proceeding to the next compilation stage.

**Figure 62. Plan Stage Periphery Placement Message**



1. In the Compilation Dashboard, click the **Plan** stage.
2. In the Compilation Report, under the **Plan Stage** folder, click the **Input Pins**, **Output Pins**, **I/O Bank Usage**, **PLL Usage Summary**, or other reports. Verify attributes of the I/O pins, such as the physical pin location, I/O standards, and PLL placement.

**Figure 63. Input Pins Report**

| Input Pins                                          |        |       |          |              |              |              |                       |
|-----------------------------------------------------|--------|-------|----------|--------------|--------------|--------------|-----------------------|
| <input type="text" value="&lt;&lt;Filter&gt;&gt;"/> |        |       |          |              |              |              |                       |
|                                                     | Name   | Pin # | I/O Bank | X coordinate | Y coordinate | Z coordinate | Combinational Fan-Out |
| 1                                                   | accel  | AL27  | 2A       | 52           | 12           | 46           | 1                     |
| 2                                                   | clock  | H10   | 3L       | 224          | 397          | 46           | 19                    |
| 3                                                   | dir[0] | AN27  | 2A       | 52           | 8            | 46           | 1                     |
| 4                                                   | dir[1] | AP30  | 2A       | 52           | 8            | 61           | 1                     |
| 5                                                   | enable | AK27  | 2A       | 52           | 6            | 46           | 8                     |
| 6                                                   | reset  | AY24  | 2A       | 52           | 4            | 16           | 6                     |





Figure 64. PLL Usage Summary Report

| PLL Usage Summary                                   |                                                      |
|-----------------------------------------------------|------------------------------------------------------|
| <input type="text" value="&lt;&lt;Filter&gt;&gt;"/> |                                                      |
| 1                                                   | u_pll jopll_0 altera_iopll_ijtwentynm_pll jopll_inst |
| 1                                                   | -- PLL Location: IOPLL_3H                            |
| 2                                                   | -- PLL Bandwidth: low                                |
| 1                                                   | -- PLL Bandwidth Range: 2470000 to 1520000 Hz        |
| 3                                                   | -- Reference Clock Frequency: 100.0 MHz              |
| 4                                                   | -- PLL VCO Frequency: 600.0 MHz                      |
| 5                                                   | -- PLL Operation Mode: direct                        |
| 6                                                   | -- PLL Freq Min Lock: 100.000000 MHz                 |
| 7                                                   | -- PLL Freq Max Lock: 208.333333 MHz                 |
| 8                                                   | -- PLL Enable: On                                    |
| 9                                                   | -- M Counter: 6                                      |

- For Intel Arria 10 and Intel Cyclone 10 designs, click **Global & Other Fast Signals Summary** report to verify which clocks the Compiler promotes to global clocks. Clock planning occurs after the Early Place stage for Intel Stratix 10 designs.

Figure 65. Global & Other Fast Signals Report Shows Clock Promotion (Intel Arria 10 and Intel Cyclone 10)

| Global & Other Fast Signals Summary                 |      |          |         |             |                |                      |
|-----------------------------------------------------|------|----------|---------|-------------|----------------|----------------------|
| <input type="text" value="&lt;&lt;Filter&gt;&gt;"/> |      |          |         |             |                |                      |
|                                                     | Name | Location | Fan-Out | Signal Type | Promotion Type | Global Resource Used |
| 1                                                   | clk  | PIN_W15  | 1320    | Global      | Automatic      | Global Clock Region  |

### 6.4.3.3 Clock Planning after Early Place (Intel Stratix 10 only)

Intel Stratix 10 devices support clock planning after the Early Place stage, rather than after the Plan stage. After running Early Place, view the Global & Other Fast Signals report to view details and plan the clocks in your project. To view clock details after Early Place, follow these steps:

- In the Compilation Dashboard, click the **Early Place** stage.
- In the Compilation Report, under the **Early Place Stage** folder, click the **Global & Other Fast Signals Details** or **Global & Other Fast Signals Summary** report.

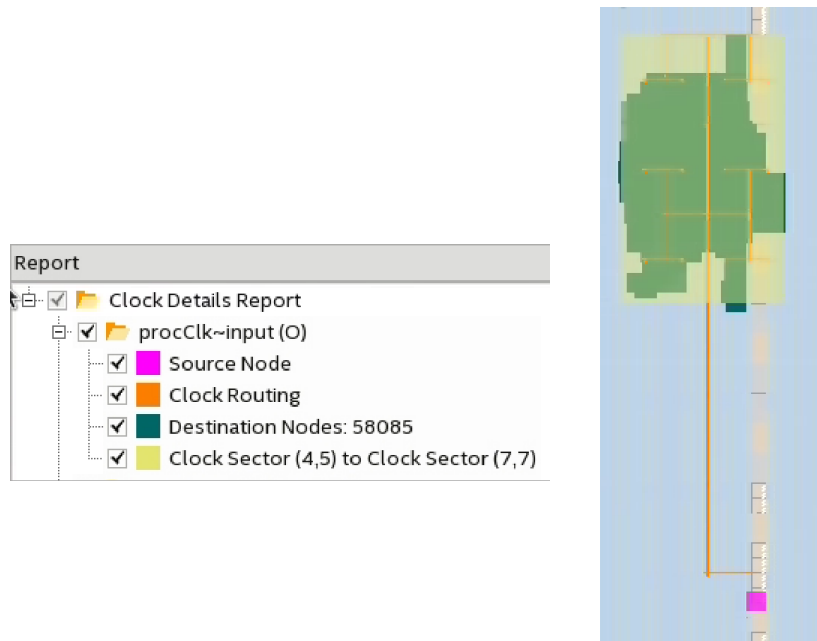
Figure 66. Global & Other Fast Signal Details Report

| Global & Other Fast Signals Details                 |                                                |                                    |
|-----------------------------------------------------|------------------------------------------------|------------------------------------|
| <input type="text" value="&lt;&lt;Filter&gt;&gt;"/> |                                                |                                    |
|                                                     | Property                                       | Value                              |
| 1                                                   | Name                                           | clock                              |
| 1                                                   | -- Source Type                                 | I/O pad                            |
| 2                                                   | -- Source Location                             | PIN_H10                            |
| 3                                                   | -- Fan-Out                                     | 19                                 |
| 4                                                   | -- Promotion Reason                            | Promoted automatically by compiler |
| 5                                                   | -- Clock Region                                | Sectors (4, 0) to (7, 1)           |
| 6                                                   | -- Clock Region Size (in Sectors)              | 4 x 2 (8 total)                    |
| 7                                                   | -- Spine Index used in each Sector             | 0                                  |
| 8                                                   | -- Path Length from Clock Source to Clock Tree | 12 clock sector wire(s)*           |
| 9                                                   | -- Clock Tree Depth                            | 1.5 clock sector wire(s)           |

The report provides clock tree path length and depth. The shortest path length from clock source to clock tree, and the smallest clock tree depth, results in the best clock performance.

3. To visualize the clock path length and clock tree depth, click **Tools > Chip Planner**.
4. In the Chip Planner **Tasks** pane, click **Report Clock Details** under the **Clock Reports** folder. The **Report** pane lists all the clocks in the design.
5. In the Report pane, select one or more clocks to highlight the clock elements in Chip Planner.

**Figure 67. Visualizing Clocks in Chip Planner**



After running the Early Place stage, you can run `late_place`, rather than the full Place stage, to reduce total compilation time.

#### 6.4.3.4 Identifying High Fan-Out Signals after Early Place

High fan-out signals increase placement difficulty. After Early Place, identify and consider moving high fan-out signals to global resources.

1. In the Compilation Dashboard, click the **Early Place** stage.
2. In the Compilation Report, under the **Early Place Stage** folder, click the **Non-Global High Fan-Out Signals** report. The report lists the number of fan-outs for each signal.

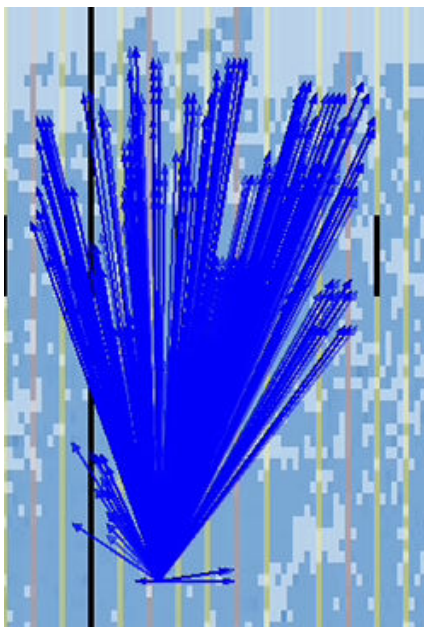


Figure 68. Non-Global High Fan-Out Signals Report

| Non-Global High Fan-Out Signals                 |                                        |         |
|-------------------------------------------------|----------------------------------------|---------|
| <input type="text" value="&lt;&lt;Filter&gt;"/> |                                        |         |
|                                                 | Name                                   | Fan-Out |
| 1                                               | uBP regs_inst registersLocal[0][96][0] | 2965    |

- To visualize the clock fan-out, right-click the signal name in the report, and then click **Locate Node** > **Locate in Chip Planner**.

Figure 69. Non-Global High Fan-Out Signal in Chip Planner



- To place those high fan-out signals on global resources, click **Assignments** > **Assignment Editor**, and then assign the high fan-out signal to a global signal before re-starting compilation.

Figure 70. Assigning High Fan-Out Signal in Assignment Editor

| To                                                                  | Assignment Name | Value        |
|---------------------------------------------------------------------|-----------------|--------------|
| <input type="text" value="uBP"/> <input type="text" value="To"/>    | Partition       | BPCalculator |
| <input type="text" value="uBP regs_inst registersLocal[0][96][0]"/> | Global Signal   | Global Clock |
| <<new>>                                                             | <<new>>         |              |

### 6.4.4 Enabling Physical Synthesis Optimization

Physical synthesis optimization improves circuit performance by performing combinational and sequential optimization and register duplication.

To enable physical synthesis options, follow these steps:

1. Click **Assignments** ► **Settings** ► **Compiler Settings**.
2. To enable retiming, combinational optimization, and register duplication, click **Advanced Settings (Fitter)**. Next, enable **Physical Synthesis**.
3. View physical synthesis results in the **Netlist Optimizations** report.

## 6.4.5 Viewing Fitter Reports

The Fitter generates detailed reports and messages for each stage of place and route. The Fitter Summary reports basic information about the Fitter run, such as date, software version, device family, timing model, and logic utilization.

### 6.4.5.1 Plan Stage Reports

The Plan stage reports describe the I/O, interface, and control signals discovered during the periphery planning stage of the Fitter.

**Figure 71. Plan Stage Reports (Intel Arria 10 and Intel Cyclone 10 GX Designs)**

The screenshot shows the 'Compilation Report - chiptrip\_nf' window. The 'Table of Contents' on the left lists various reports, with 'Input Pins' selected. The main window displays a table of input pins with the following data:

|   | Name   | Pin # | I/O Bank | X coordinate | Y coordinate | Z coord |
|---|--------|-------|----------|--------------|--------------|---------|
| 1 | accel  | AH27  | 2F       | 78           | 35           | 31      |
| 2 | clock  | A21   | 2L       | 78           | 196          | 46      |
| 3 | dir[0] | AP30  | 2F       | 78           | 39           | 46      |
| 4 | dir[1] | AF26  | 2F       | 78           | 39           | 61      |
| 5 | enable | AR29  | 2F       | 78           | 37           | 46      |
| 6 | reset  | AN26  | 2F       | 78           | 38           | 31      |

For Intel Arria 10 and Intel Cyclone 10 designs, the Plan stage includes the **Global & Other Fast Signals Summary** report that allows you to verify which clocks the Compiler promotes to global clocks. Clock planning occurs after the Early Place stage for Intel Stratix 10 designs.

### 6.4.5.2 Early Place Stage Reports

During Early Place the Fitter begins assigning core design logic to device resources. For Intel Stratix 10 designs, the Early Place stage reports include the **Global & Other Fast Signals Summary** and **Global & Other Fast Signals Details** reports. Use these reports to verify which clocks the Compiler promotes to global clocks. Clock planning occurs after the Plan stage for Intel Arria 10 and Intel Cyclone 10 designs.



Figure 72. Early Place Stage Reports (Intel Stratix 10 Design)

| Resource |                                                   | Usage        |
|----------|---------------------------------------------------|--------------|
| 1        | -- Logic utilization (ALM...total ALMs on device) | 11 / 933,120 |
| 2        | [-] ALMs needed [=A-B+C]                          | 11           |
| 1        | [-] [A] ALMs used in f...acement [=a+b+c+d]       | 12 / 933,120 |
| 1        | -- [a] ALMs used for ...register circuitry        | 9            |
| 2        | -- [b] ALMs used for LUT logic                    | 2            |
| 3        | -- [c] ALMs used for register circuitry           | 1            |
| 4        | -- [d] ALMs used for ...alf of total ALMs)        | 0            |
| 2        | -- [B] Estimate of AL...e by dense packing        | 1 / 933,120  |
| 3        | [-] [C] Estimate of ALM...available [=a+b+c+d]    | 0 / 933,120  |
| 1        | -- [a] Due to location constrained logic          | 0            |
| 2        | -- [b] Due to LAB-wide signal conflicts           | 0            |
| 3        | -- [c] Due to LAB input limits                    | 0            |
| 4        | -- [d] Due to virtual I/Os                        | 0            |
| 3        | --                                                |              |
| 4        | -- Difficulty packing design                      | N/A          |
| 5        | --                                                |              |

### 6.4.5.3 Place Stage Reports

The Place stage reports describe all device resources the Fitter allocates during logic placement. The report details include the type, number, and overall percentage of each resource type.

Figure 73. Place Stage Reports

| Compilation Hierarchy Node | ALMs needed [=A-B+C] | [A] ALMs   |
|----------------------------|----------------------|------------|
| 1 [-]                      | 37.5 (0.0)           | 14.5 (0.0) |
| 1  auto                    | 9.5 (9.5)            | 5.5 (5.5)  |
| 2  speed                   | 9.5 (9.5)            | 2.0 (2.0)  |
| 3  tick                    | 9.5 (9.5)            | 3.0 (3.0)  |
| 4  time_c                  | 9.0 (9.0)            | 4.0 (4.0)  |

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses

### 6.4.5.4 Route Stage Reports

The Route stage reports describe all device resources that the Fitter allocates during routing. Details include the type, number, and overall percentage of each resource type. The Route stage also reports delay chain summary information.

Figure 74. Route Stage Reports

|    | Name      | Pin Type | Input Delay Chain | Output Delay Chain |
|----|-----------|----------|-------------------|--------------------|
| 1  | accel     | Input    | 0                 | --                 |
| 2  | at_altera | Output   | --                | 0                  |
| 3  | clock     | Input    | 0                 | --                 |
| 4  | dir[0]    | Input    | 0                 | --                 |
| 5  | dir[1]    | Input    | 0                 | --                 |
| 6  | enable    | Input    | 0                 | --                 |
| 7  | gt1       | Output   | --                | 0                  |
| 8  | gt2       | Output   | --                | 0                  |
| 9  | reset     | Input    | 0                 | --                 |
| 10 | stf       | Output   | --                | 0                  |
| 11 | ticket[0] | Output   | --                | 0                  |
| 12 | ticket[1] | Output   | --                | 0                  |

### 6.4.5.5 Retime Stage Reports

The Fitter generates detailed reports showing the results of the Retime stage, including the Retiming Limit Details report. This report lists hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths.

Figure 75. Retiming Limit Details

|   | Clock Transfer   | Limiting Reason        | Recommendation                                      |
|---|------------------|------------------------|-----------------------------------------------------|
| 1 | Clock Domain clk | Insufficient Registers | See the Fast Forward Timing Closure R...RTL change: |

|   | Path Info            | Register                | Element                                 |
|---|----------------------|-------------------------|-----------------------------------------|
| 1 | Retiming Restriction | REG                     | #1 window_contol window_column_counter[ |
| 2 | Long Path (Critical) |                         | window_contol window_column_counter[    |
| 3 | Long Path (Critical) |                         | window_contol window_column_cou...GOR   |
| 4 | Long Path (Critical) |                         | window_contol window_column_cou...L_IN  |
| 5 | Long Path (Critical) | Bypassed Hyper-Register | window_contol window_column_cou...eg0   |
| 6 | Long Path (Critical) |                         | window_contol reduce_or_0 datac         |
| 7 | Long Path (Critical) |                         | window_contol reduce_or_0 combout       |
| 8 | Long Path (Critical) |                         | window_contol reduce_or_0~la_lab/labou  |

### 6.4.5.6 Finalize Stage Reports

The Finalize stage reports describe final placement and routing operations, including:

- HSLP Summary. For Intel Arria 10 designs, the Compiler converts unnecessary tiles to High-Speed or Low-Power (HSLP) tiles.
- Post-route hold fix-up data. For Intel Stratix 10 designs, the Compiler reports hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths.

Figure 76. Finalize Stage Reports (Intel Stratix 10 Design)

|    | Routing Resource Type         | Usage                  |
|----|-------------------------------|------------------------|
| 1  | Block Input Muxes             | 16 / 675,444 (< 1 %)   |
| 2  | Block interconnects           | 57 / 9,132,912 (< 1 %) |
| 3  | C16 interconnects             | 35 / 226,512 (< 1 %)   |
| 4  | C2 interconnects              | 19 / 1,359,072 (< 1 %) |
| 5  | C3 interconnects              | 34 / 2,758,032 (< 1 %) |
| 6  | C4 interconnects              | 17 / 1,772,208 (< 1 %) |
| 7  | CLOCK_INVERTs                 | 0 / 7,616 (0 %)        |
| 8  | DCM_muxes                     | 1 / 1,632 (< 1 %)      |
| 9  | Direct links                  | 19 / 9,132,912 (< 1 %) |
| 10 | GAP Interconnects             | 40 / 267,192 (< 1 %)   |
| 11 | GAPs                          | 0 / 29,304 (0 %)       |
| 12 | HIO Buffers                   | 5 / 209,664 (< 1 %)    |
| 13 | Horizontal Buffers            | 5 / 176,364 (< 1 %)    |
| 14 | Horizontal_cl...segment_muxes | 6 / 7,488 (< 1 %)      |
| 15 | Programmable Inverts          | 16 / 318,960 (< 1 %)   |
| 16 | R10 interconnects             | 96 / 2,456,208 (< 1 %) |
| 17 | R2 interconnects              | 23 / 2,265,120 (< 1 %) |
| 18 | R24 interconnects             | 69 / 293,040 (< 1 %)   |
| 19 | R24/C16 interconnect drivers  | 44 / 453,024 (< 1 %)   |
| 20 | R4 interconnects              | 28 / 3,248,028 (< 1 %) |
| 21 | Row Clock Tap-Offs            | 4 / 725,544 (< 1 %)    |
| 22 | Switchbox_clock_muxes         | 53 / 41,600 (< 1 %)    |
| 23 | Vertical_seam_tap_muxes       | 40 / 22,848 (< 1 %)    |

**Related Links**

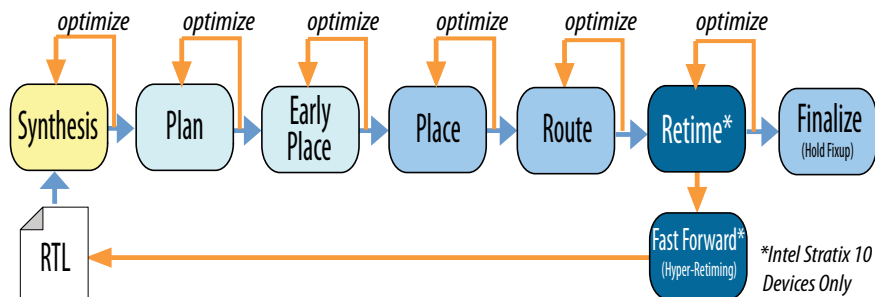
Step 2: Review Retiming Results on page 211  
For information on Retiming and Fast Forward compilation reports

**6.5 Running the Hyper-Aware Design Flow**

The Intel Quartus Prime Pro Edition Compiler helps you to take full advantage of the Intel Stratix 10 Intel Hyperflex architecture. Use the Hyper-Aware design flow to shorten design cycles and optimize performance.

The Hyper-Aware design flow combines automated register retiming (Hyper-Retiming), with implementation of targeted timing closure recommendations (Fast Forward compilation), to maximize use of Hyper-Registers and drive the highest performance for Intel Stratix 10 designs.

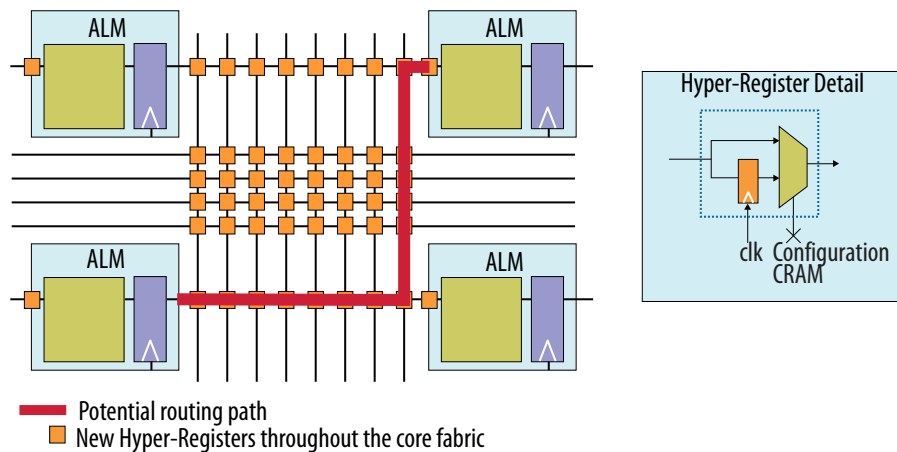
Figure 77. Hyper-Aware Design Flow



## Hyper-Retiming

A key innovation of the Intel Stratix 10 architecture is the addition of multiple Hyper-Registers in every routing segment and block input. Maximizing the use of Hyper-Registers improves design performance. The prevalence of Hyper-Registers improves balance of time delays between registers and mitigates critical path delays. Hyper-Retiming moves registers out of ALMs and retimes them into Hyper-Registers, wherever advantageous. Hyper-Retiming runs automatically during fitting, requires minimal effort, and can result in significant performance improvement.

**Figure 78. Hyper-Register Architecture**



## Fast Forward Compilation

If you require optimization beyond Hyper-Retiming, run Fast Forward compilation to generate timing closure recommendations that break key performance bottlenecks. Fast Forward compilation shows precisely where to make the most impact with RTL changes, and reports the performance benefits you can expect from each change. The Fitter does not automatically retime registers across RAM and DSP blocks. However, Fast Forward analysis shows the potential performance benefit from this optimization.

Fast-Forward compilation identifies the best location to add pipeline stages (Hyper-Pipelining), and the expected performance benefit in each case. After you modify the RTL to place pipeline stages at the boundaries of each clock domain, the Hyper-Retimer automatically places the registers within the clock domain at the optimal locations to maximize performance. Implement the recommendations in RTL to achieve similar results. After implementing any changes, re-run the Hyper-Retimer until the results meet performance and timing requirements. Fast Forward compilation does not run automatically as part of a full compilation. Enable or run **Fast Forward compilation** in the Compilation Dashboard.





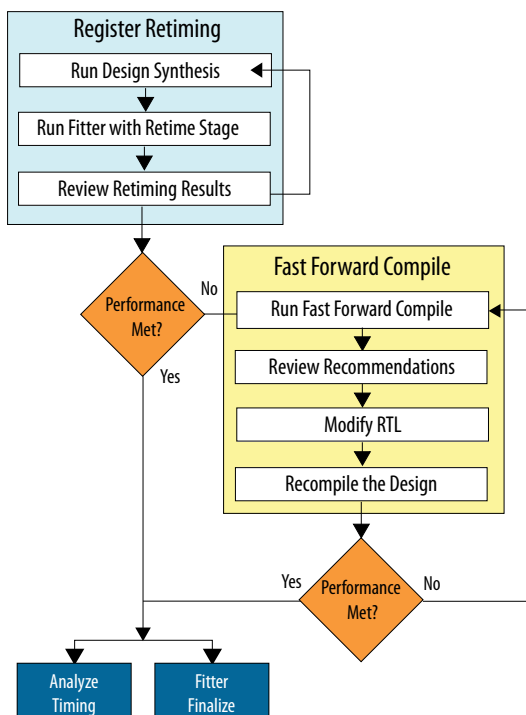
**Table 40. HyperFlex Optimization Steps**

| Optimization Step | Technique            | Description                                                                                                                                                          |
|-------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Step 1            | Hyper-Retiming       | Retimer moves existing registers into Hyper-Registers.                                                                                                               |
| Step 2            | Fast Forward Compile | Compiler generates design-specific timing closure recommendations and predicts performance improvement.                                                              |
| Step 3            | Hyper-Pipelining     | Use Fast Forward compilation to identify where to add new registers and pipeline stages in RTL.                                                                      |
| Step 4            | Hyper-Optimization   | Design optimization beyond Hyper-Retiming and Hyper-Pipelining, such as restructuring loops, removing control logic limits, and reducing the delay along long paths. |

The Hyper-Aware design flow includes the following high-level steps this chapter covers in detail:

1. Run the Retime stage during the Fitter to automatically retime ALM registers into Hyper-Registers.
2. Review Retiming Results in the Compilation Report.
3. If you require further performance optimization, run Fast Forward compilation.
4. Review Fast Forward timing closure recommendations.
5. Implement appropriate Fast Forward recommendations in your RTL.
6. Recompile the design through the Retime stage.

**Figure 79. Hyper-Aware Design Flow**



### 6.5.1 Step 1: Run Register Retiming

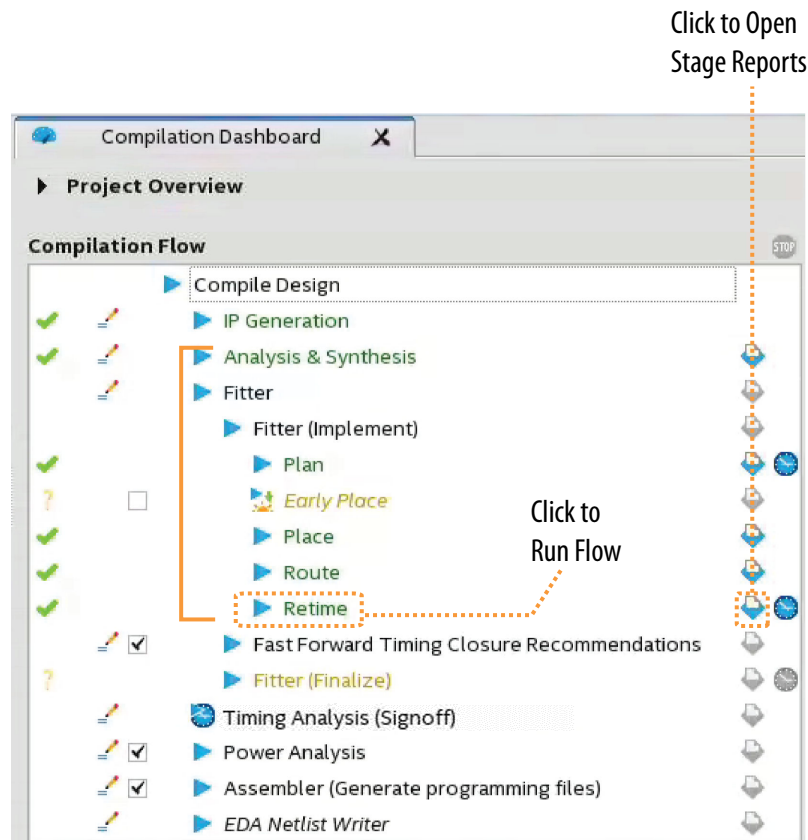
Register retiming improves design performance by moving registers out of ALMs and retimes them into Hyper-Registers in the Intel Stratix 10 device interconnect.

The Fitter runs the **Retime** stage automatically following place and route when you target an Intel Stratix 10 device. Alternatively, start or stop the individual **Retime** stage in the Compilation Dashboard. After running register retiming, view the Fitter reports to optimize remaining critical paths.

To run register retiming:

1. Create or open an Intel Quartus Prime project that is ready for design synthesis and fitting.
2. To run register retiming, click **Retime** on the Compilation Dashboard. The Compiler runs prerequisite stages automatically, as needed. The Compiler generates detailed reports and timing analysis data for each stage. Click the **Report** or **Timing Analyzer** icons to review results of each stage. Rerun any stage to apply any setting or design changes.
3. If register retiming achieves all performance goals for your design, proceed to Fitter (Finalize) and timing analysis stages of compilation. If your design requires further optimization, run **Fast Forward Timing Closure Recommendations**.

**Figure 80. Retiming Stage in Compilation Dashboard**





## 6.5.2 Step 2: Review Retiming Results

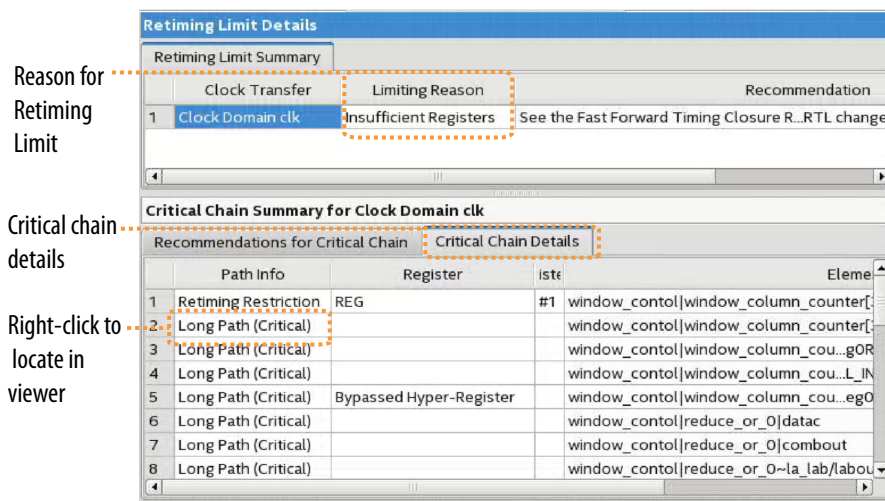
The Fitter generates detailed reports showing the results of the Retime stage. Follow these steps to review the results and make additional performance improvements with register retiming.

1. To open the **Retiming Limit Details** report, click the **Report** icon for the Retime stage in the Compilation Dashboard. The **Retiming Limit Details** lists the number of registers moved, their paths, and the limiting reason preventing further retiming.
2. To further optimize, resolve any **Limiting Reason** in your design, and then rerun the **Retime** stage, as necessary.
3. If register retiming achieves all performance goals for your design, proceed to Fitter (Finalize) and Timing Analysis stages of compilation.
4. If your design requires further optimization, run **Fast Forward Timing Closure Recommendations**.

**Table 41. Retiming Limit Details Report Data**

| Report Data                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Clock Transfer</b>         | Lists each clock domain in your design. Click the domain to display data about each entry.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Limiting Reason</b>        | Specifies any design condition that prevent further register retiming improvement, such as any of the following conditions: <ul style="list-style-type: none"> <li>• <b>Insufficient Registers</b>—indicates insufficient quantity of registers at either end of the chain for retiming. Adding more registers can improve performance.</li> <li>• <b>Short Path/Long Path</b>—indicates that the critical chain has dependent paths with conflicting characteristics. For example, one path improves performance with more registers, and another path has no place for additional registers.</li> <li>• <b>Path Limit</b>—indicates that there are no further Hyper-Register locations available on the critical path, or the design reached a performance limit of the current place and route.</li> <li>• <b>Loops</b>—indicates a feedback path in a circuit. When the critical chain includes a feedback loop, retiming cannot change the number of registers in the loop without changing functionality. The Compiler can retime around the loop without changing functionality. However, the Compiler cannot place additional registers in the loop.</li> </ul> |
| <b>Critical Chain Details</b> | Lists register timing path associated with the retiming limitations. Right-click any path to <b>Locate Critical Chain in Technology Map Viewer</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

**Figure 81. Retiming Limit Details**



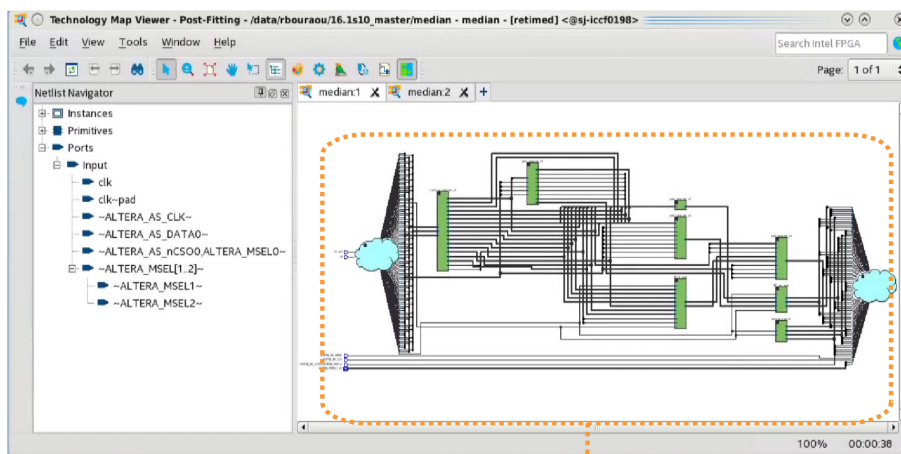
*Note:* The Compiler reports any hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths.

### 6.5.2.1 Locate Critical Chains

The **Retiming Limit Details** reports the design paths that limit further register retiming. Right-click any path to locate to the path in the Technology Map Viewer - Post-fitting view. This viewer displays a schematic representation of the complete design after place, route, and register retiming. To view the retimed netlist in the Technology Map Viewer, follow these steps:

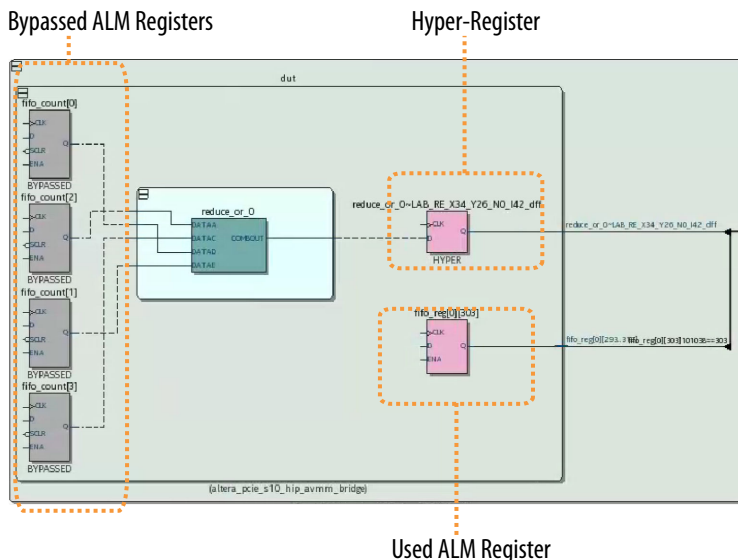
1. To open the **Retiming Limit Details** report, click the **Report** icon next to the **Retime** stage in the Compilation Dashboard.
2. Right-click any path in the **Retiming Limit Details** report and click **Locate Critical Chain in Technology Map Viewer**. The netlist displays as a schematic in the Technology Map Viewer.

**Figure 82. Technology Map Viewer**



Schematic View of Design Netlist

Figure 83. Post-Fit Viewer After Retiming



### 6.5.3 Step 3: Run Fast Forward Compile and Hyper-Retiming

When you run Fast Forward compilation, the Compiler predictively removes signals from registers to allow mobility within in the netlist for subsequent retiming. Fast Forward compilation generates design-specific timing closure recommendations, and predicts maximum performance with removal of all timing restrictions. After you complete Fast Forward explorations, determine which recommendations you can implement to provide the most benefit. Implement appropriate recommendations in your RTL, and recompile the design to realize the performance levels that Fast Forward reports.

To generate Fast Forward timing closure recommendations, follow these steps:

1. On the Compilation Dashboard, click **Fast Forward Timing Closure Recommendations**. The Compiler runs prerequisite synthesis or Fitter stages automatically, as needed, and generates timing closure recommendations in the Compilation Report.
2. View timing closure recommendations in the Compilation Report to evaluate design performance and implement key RTL performance improvements.
3. Optionally, specify any of the following any of the following options if you want to automate or refine Fast Forward analysis:
  - If you want to run Fast Forward compilation during each full compilation, click **Assignments > Settings > Compiler Settings > HyperFlex** and enable **Run Fast Forward Timing Closure Recommendations during compilation**.
  - If you want to modify how Fast Forward compilation interprets specific I/O and block types, click **Assignments > Settings > Compiler Settings > HyperFlex > Advanced Settings**.

Figure 84. Running Fast Forward Compilation

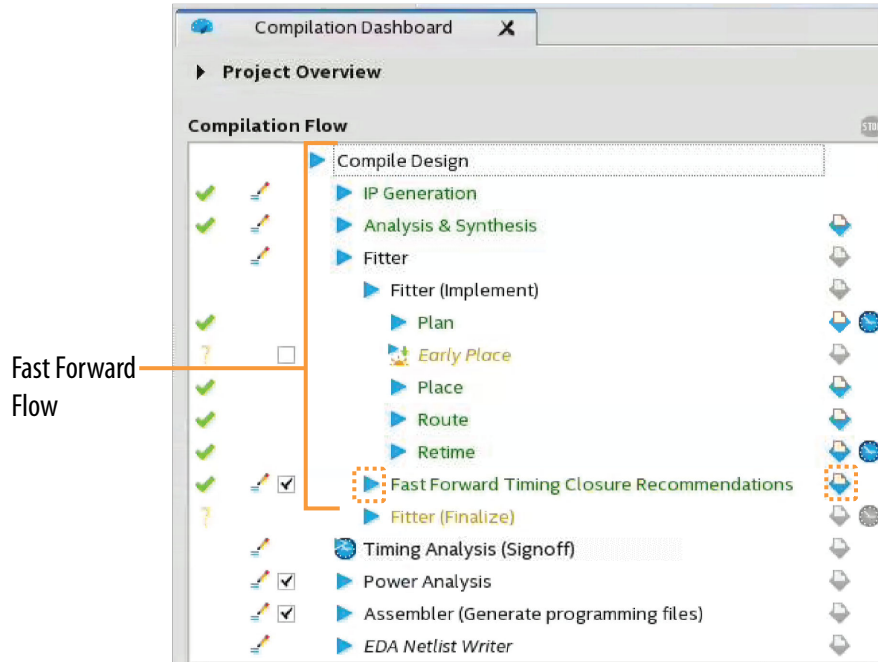
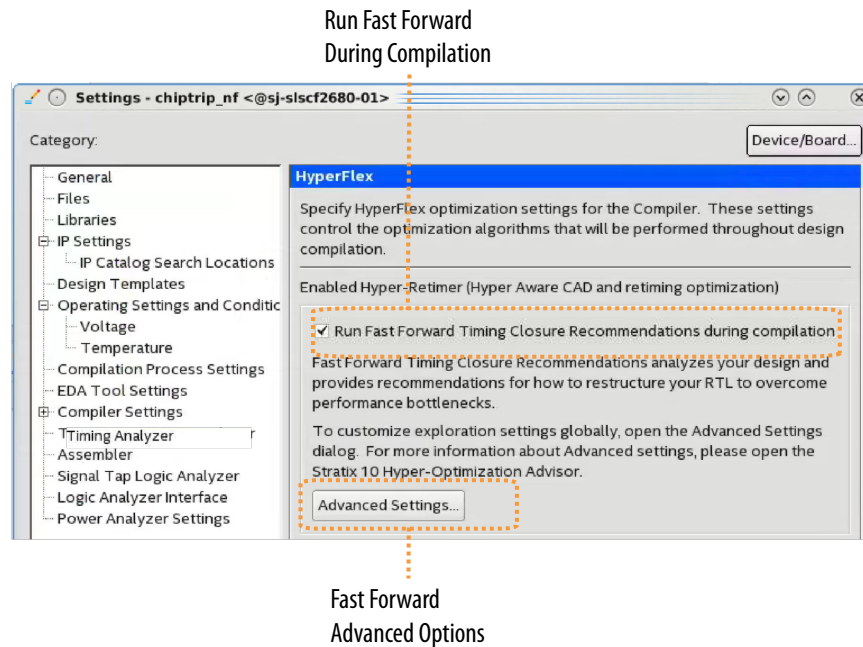


Figure 85. HyperFlex Settings





### 6.5.3.1 Advanced HyperFlex Settings

The **Advanced HyperFlex Settings** control how Fast Forward Compilation analyzes and reports results for specific logical structures in the Intel Hyperflex architecture of the Intel Stratix 10 FPGA. To access the settings, click **Assignments > Settings > HyperFlex > Advanced Settings**.

**Table 42. Advanced HyperFlex Settings**

| Option                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Fast Forward Compile Asynchronous Clears</b>                | Specifies how Fast Forward analysis accounts for registers with asynchronous clear signals. The options are: <ul style="list-style-type: none"> <li><b>Auto</b>—the Compiler identifies asynchronous clears as asynchronous until they limit timing performance during Fast Forward Compilation, at which point the Compiler identifies the asynchronous clears as removed.</li> <li><b>Preserve</b>—the Compiler never assumes removal or conversion of asynchronous clears for Fast Forward analysis.</li> </ul> |
| <b>Fast Forward Compile Fully Registered DSP Blocks</b>        | Specifies how Fast Forward analysis accounts for DSP blocks that limit performance. Enable this option to generate results as if all DSP blocks are fully registered.                                                                                                                                                                                                                                                                                                                                              |
| <b>Fast Forward Compile Fully Registered RAM Blocks</b>        | Specifies how Fast Forward analysis accounts for RAM blocks that limit performance. Enable this option to analyze the blocks as fully registered.                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Fast Forward Compile Maximum Additional Pipeline Stages</b> | Specifies the maximum number of pipeline stages that Fast Forward compilation explores.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Fast Forward Compile User Preserve Directives</b>           | Specifies how Fast Forward compilation accounts for restrictions from user-preserve directives.                                                                                                                                                                                                                                                                                                                                                                                                                    |

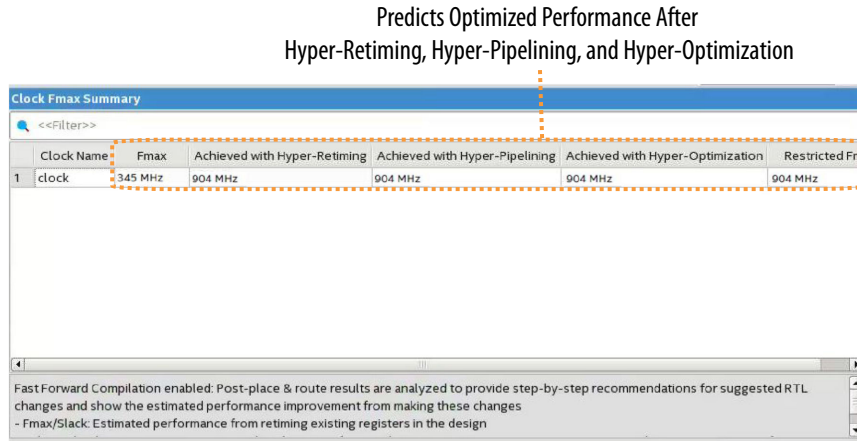
### 6.5.4 Step 4: Review Hyper-Retiming Results

After running Fast Forward Compilation, review the reports to determine which recommendations are appropriate and practical for your design functionality and performance goals.

#### 6.5.4.1 Clock Fmax Summary Report

The Clock Fmax Summary reports the current  $f_{max}$  and potential performance achievable for each clock domain after Hyper-Retiming, Hyper-Pipelining, and Hyper-Optimization steps. Review the Clock Fmax Summary data to determine whether each potential performance improvement warrants further investigation and potential optimization of design RTL.

Figure 86. Current and Potential Performance in Clock Fmax Summary



### 6.5.4.2 Fast Forward Details Report

The Fast Forward Details report recommends the design modifications necessary to achieve Fast Forward compilation performance levels. Some recommendations may be functionally impossible or impractical for your design. Consider which recommendations you can implement in RTL to achieve similar performance improvement. Click any optimization **Step** to view the implementation details and performance calculations for that step.

Table 43. Fast Forward Details Report Data

| Report Field                                        | Description                                                                                                                                                                                                                                                     |
|-----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Step</b>                                         | Displays the pre-optimized Base Performance $f_{MAX}$ , the recommended Fast Forward optimization steps, and the Fast Forward Limit critical path that prevents further optimization.                                                                           |
| <b>Fast Forward Optimizations Analyzed</b>          | Summarizes the optimizations necessary to implement each optimization step.                                                                                                                                                                                     |
| <b>Estimated Fmax</b>                               | Specifies the potential $f_{MAX}$ performance if you implement all Fast Forward optimization steps.                                                                                                                                                             |
| <b>Optimizations Analyzed For Fast Forward Step</b> | Lists design recommendations hierarchically for the selected <b>Step</b> . Click the text to expand the report and view the clock domain, the affected module, and the bus and bits that require modification.                                                  |
| <b>Optimizations Analyzed (Cumulative)</b>          | Accumulated list of all design changes necessary to reach the selected <b>Step</b> .                                                                                                                                                                            |
| <b>Critical Chain at Fast Forward Limit</b>         | Displays information about any path that continues to limit Hyper-Retiming even after application of all Fast Forward steps. The critical chain is any path that limits further Hyper-Retiming. Click the <b>Fast Forward Limit</b> step to display this field. |
| <b>Recommendations for Critical Chain</b>           | Lists register timing path associated with the retiming limitations. Right-click any path to <b>Locate Critical Chain in Fast Forward Viewer</b> .                                                                                                              |





Figure 87. Fast-Forward Details Report

| Fast Forward Details for Clock Domain clk |                                       |                                             |                |
|-------------------------------------------|---------------------------------------|---------------------------------------------|----------------|
| Fast Forward Summary for Clock Domain clk |                                       | Fast Forward Limit Critical Chain Schematic |                |
| Step                                      |                                       | Fast Forward Optimizations Analyzed         | Estimated Fmax |
| 1                                         | Base Performance                      | None                                        | 758 MHz        |
| 2                                         | Fast Forward Step #1 (Hyper-Retiming) | Removed asynchronous clears on 22 Registers | 994 MHz        |
| 3                                         | Fast Forward Limit                    | Performance Limited by: RTL Loop            | --             |

| Optimizations Analyzed for Fast Forward Step 1 (994 MHz) |                                                                                  |
|----------------------------------------------------------|----------------------------------------------------------------------------------|
| Optimizations Analyzed (Cumulative)                      |                                                                                  |
|                                                          | Optimizations Analyzed (Cumulative)                                              |
| 1                                                        | Removed asynchronous clears on 22 Registers (1 Domain)                           |
| 1                                                        | Removed asynchronous clears on 22 Registers in Clock Domain 'clk' (1 Entity)     |
| 1                                                        | Removed asynchronous clears on 22 Registers in Entity state_machine (1 Instance) |
| 1                                                        | Removed asynchronous clears on 22 Registers in Instance window_control           |

Right-click any path to locate to the critical chain in the Fast Forward Viewer. The Fast Forward Viewer displays a predictive representation of the complete design, after implementation of all Fast Forward recommendations.

Figure 88. Recommendations for Critical Chain

| Fast Forward Details for Clock Domain clk |                                       |                                             |                |
|-------------------------------------------|---------------------------------------|---------------------------------------------|----------------|
| Fast Forward Summary for Clock Domain clk |                                       | Fast Forward Limit Critical Chain Schematic |                |
| Step                                      |                                       | Fast Forward Optimizations Analyzed         | Estimated Fmax |
| 1                                         | Base Performance                      | None                                        | 758 MHz        |
| 2                                         | Fast Forward Step #1 (Hyper-Retiming) | Removed asynchronous clears on 22 Registers | 994 MHz        |
| 3                                         | Fast Forward Limit                    | Performance Limited by: RTL Loop            | --             |

| Critical Chain at Fast Forward Limit |                                                                                                                                                                                                                                                                                     |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Optimizations Analyzed (Cumulative)  |                                                                                                                                                                                                                                                                                     |
|                                      | Recommendations for Critical Chain                                                                                                                                                                                                                                                  |
| Recommendation                       |                                                                                                                                                                                                                                                                                     |
| 1                                    | The critical chain is limited by: RTL Loop                                                                                                                                                                                                                                          |
| 2                                    |                                                                                                                                                                                                                                                                                     |
| 3                                    | 1) Make RTL design changes as described in the 'Optimizations Analyzed (Cumulative)' table                                                                                                                                                                                          |
| 4                                    | 2) The critical chain has a RTL loop that restricts retiming.<br>Reduce the delay of 'Long Paths' in the loop<br>or insert more pipeline stages in 'Long Paths' in the loop<br>or see the Stratix 10 High Performance Design Handbook for information about restructuring RTL loops |

Implement Fast Forward Timing Closure Recommendations

Figure 89. Locate Critical Chain in Fast Forward Viewer

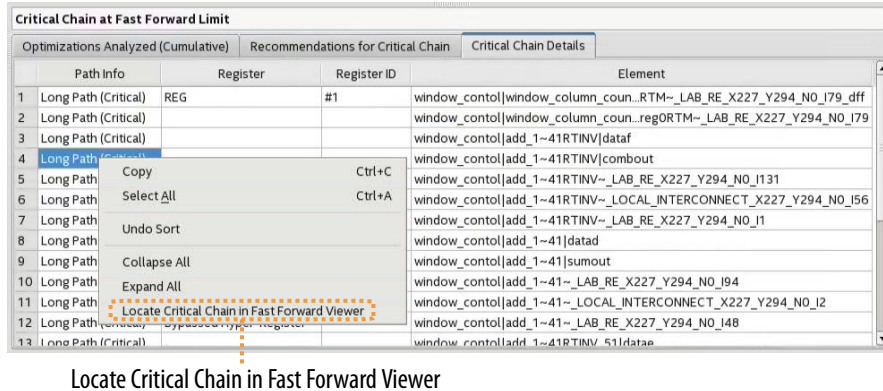
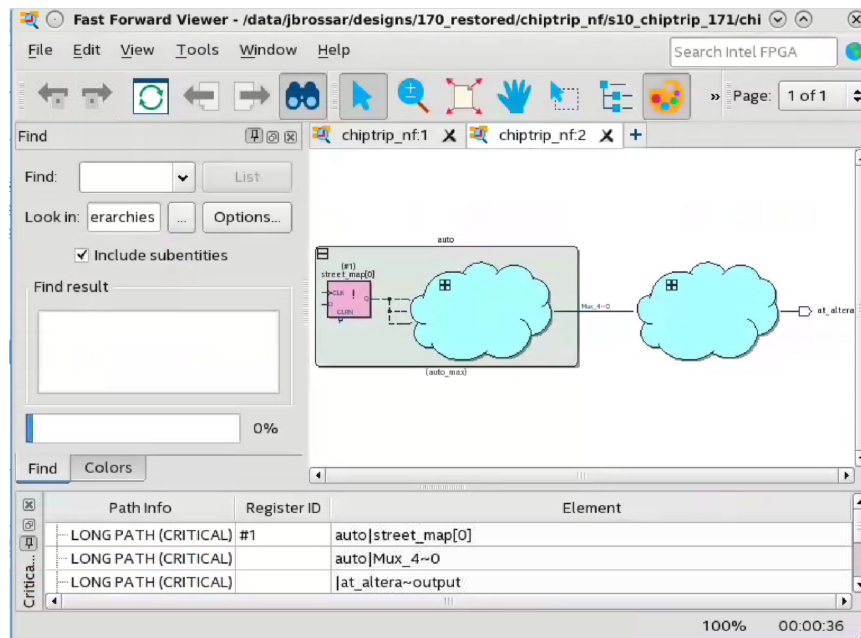


Figure 90. Fast Forward Viewer Shows Predictive Results



### 6.5.5 Step 5: Implement Fast Forward Recommendations

Implement the Fast Forward timing closure recommendations in your design RTL and rerun the **Retime** stage to realize the predictive performance gains. The amount and type of changes that you implement depends on your performance goals. For example, if you can achieve the target  $f_{MAX}$  with simple asynchronous clear removal or conversion, you can stop design optimization after making those changes. However, if you require additional performance, implement more Fast Forward recommendations, such as any of the following techniques:

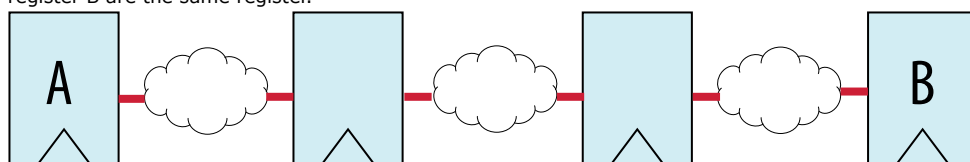
- Remove limitations of control logic, such as long feedback loops and state machines.
- Restructure logic to use functionally equivalent feed-forward or pre-compute paths, rather than long combinatorial feedback path.
- Reduce the delay of 'Long Paths' in the chain. Use standard timing closure techniques to reduce delay. Excessive combinatorial logic, sub-optimal placement, and routing congestion cause delay on paths.
- Insert more pipeline stages in 'Long Paths' in the chain. Long paths have the most delay between registers in the critical chain.
- Increase the delay (or add pipeline stages to 'Short Paths' in the chain).
- Explore performance and implement the RTL changes to your code until you reach the desired performance target.

### 6.5.5.1 Retiming Restrictions and Workarounds

The Compiler identifies the register chains in your design that limit further optimization through Hyper-Retiming. The Compiler refers to these related register-to-register paths as a critical chain. The  $f_{MAX}$  of the critical chain and its associated clock domain is limited by the average delay of a register-to-register path, and quantization delays of indivisible circuit elements like routing wires. There are a variety of situations that cause retiming restrictions. Retiming restrictions exist because of hardware characteristics, software behavior, or are inherent to the design. The **Retiming Limit Details** report the limiting reasons preventing further retiming, and the registers and combinational nodes that comprise the chain. The Fast Forward recommendations list the steps you can take to remove critical chains and enable additional register retiming.

**Figure 91. Sample Critical Chain**

In this figure the red line represents a same critical chain. Timing restrictions prevent register A from retiming forward. Timing restrictions also prevent register B from retiming backwards. A loop occurs when register A and register B are the same register.



Fast Forward recommendations for the critical chain include:

- Reduce the delay of 'Long Paths' in the chain. Use standard timing closure techniques to reduce delay. Combinational logic, sub-optimal placement, and routing congestion, are among the reasons for path delay..
- Insert more pipeline stages in 'Long Paths' in the chain. Long paths are the parts of the critical chain that have the most delay between registers.
- Increase the delay (or add pipeline stages to 'Short Paths' in the chain).



Particular registers in critical chains can limit performance for many other reasons. The Compiler classifies the following types of reasons that limit further optimization by retiming:

- Insufficient Registers
- Loop
- Short path/long path
- Path limit

After understanding why a particular critical chain limits your design's performance, you can then make RTL changes to eliminate that bottleneck and increase performance.

**Table 44. Hyper-Register Support for Various Design Conditions**

| Design Condition                                      | Hyper-Register Support                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Initial conditions that cannot be preserved           | Hyper-Registers do not have initial condition support. However, you cannot perform some retiming operations while preserving the initial condition stage of all registers (that is, the merging and duplicating of Hyper-Registers). If this condition occurs in the design, the Fitter does not retime those registers. This retiming limit ensures that the register retiming does not affect design functionality.                         |
| Register has an asynchronous clear                    | Hyper-Registers support only data and clock inputs. Hyper-Registers do not have control signals such as asynchronous clears, presets, or enables. The Fitter cannot retime any register that has an asynchronous clear. Use asynchronous clears only when necessary, such as state machines or control logic. Often, you can avoid or remove asynchronous clears from large parts of a datapath.                                              |
| Register drives an asynchronous signal                | This design condition is inherent to any design that uses asynchronous resets. Focus on reducing the number of registers that are reset with an asynchronous clear.                                                                                                                                                                                                                                                                           |
| Register has don't touch or preserve attributes       | The Compiler does not retime registers with these attributes. If you use the preserve attribute to manage register duplication for high fan-out signals, try removing the preserve attribute. The Compiler may be able to retime the high fan-out register along each of the routing paths to its destinations. Alternatively, use the dont_merge attribute. The Compiler retimes registers in ALMs, DDIOs, single port RAMs, and DSP blocks. |
| Register is a clock source                            | This design condition is uncommon, especially for performance-critical parts of a design. If this retiming restriction prevents you from achieving the required performance, consider whether a PLL can generate the clock, rather than a register.                                                                                                                                                                                           |
| Register is a partition boundary                      | This condition is inherent to any design that uses design partitions. If this retiming restriction prevents you from achieving the required performance, add additional registers inside the partition boundary for Hyper-Retiming.                                                                                                                                                                                                           |
| Register is a block type modified by an ECO operation | This restriction is uncommon. Avoid the restriction by making the functional change in the design source and recompiling, rather than performing an ECO.                                                                                                                                                                                                                                                                                      |
| Register location is an unknown block                 | This restriction is uncommon. You can often work around this condition by adding extra registers adjacent to the specified block type.                                                                                                                                                                                                                                                                                                        |
| Register is described in the RTL as a latch           | Hyper-Registers cannot implement latches. The Compiler infers latches because of RTL coding issues, such as incomplete assignments. If you do not intend to implement a latch, change the RTL.                                                                                                                                                                                                                                                |
| Register location is at an I/O boundary               | All designs contain I/O, but you can add additional pipeline stages next to the I/O boundary for Hyper-Retiming.                                                                                                                                                                                                                                                                                                                              |
| <b>continued...</b>                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                               |

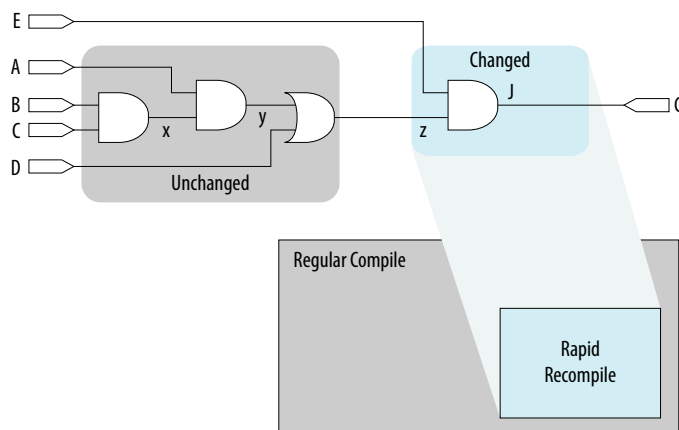
| Design Condition                                                                                              | Hyper-Register Support                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Combinational node is fed by a special source                                                                 | This condition is uncommon, especially for performance-critical parts of a design.                                                                                                                                                                                                                                                                                              |
| Register is driven by a locally routed clock                                                                  | Only the dedicated clock network clocks Hyper-Registers. Using the routing fabric to distribute clock signals is uncommon, especially for performance-critical parts of a design. Consider implementing a small clock region instead.                                                                                                                                           |
| Register is a timing exception end-point                                                                      | The Compiler does not retime registers that are sources or destinations of <code>.sdc</code> constraints.                                                                                                                                                                                                                                                                       |
| Register with inverted input or output                                                                        | This condition is uncommon.                                                                                                                                                                                                                                                                                                                                                     |
| Register is part of a synchronizer chain                                                                      | The Fitter optimizes synchronizer chains to increase the mean time between failure (MTBF), and the Compiler does not retime registers that are detected or marked as part of a synchronizer chain. Add more pipeline stages at the clock domain boundary adjacent to the synchronizer chain to provide flexibility for the retiming.                                            |
| Register with multiple period requirements for paths that start or end at the register (cross-clock boundary) | This situation occurs at any cross-clock boundary, where a register latches data on a clock at one frequency, and fans out to registers running at another frequency. The Compiler does not retime registers at cross-clock boundaries. Consider adding additional pipeline stages at one side of the clock domain boundary, or the other, to provide flexibility for retiming. |

The following topics describes RTL design techniques that you can use to remove retiming restrictions.

## 6.6 Running Rapid Recompile

During Rapid Recompile the Compiler reuses previous synthesis and fitting results whenever possible, and does not reprocess unchanged design blocks. Use Rapid Recompile to reduce timing variations and the total recompilation time after making small design changes.

**Figure 92. Rapid Recompile**



**Note:** Rapid Recompile does not support Intel Stratix 10 devices.

To run Rapid Recompile, follow these steps:

1. To start Rapid Recompile following an initial compilation (or after running the Route stage of the Fitter), click **Processing > Start > Start Rapid Recompile**. Rapid Recompile implements the following types of design changes without full recompilation:
  - Changes to nodes tapped by the Signal Tap Logic Analyzer
  - Changes to combinational logic functions
  - Changes to state machine logic (for example, new states, state transition changes)
  - Changes to signal or bus latency or addition of pipeline registers
  - Changes to coefficients of an adder or multiplier
  - Changes register packing behavior of DSP, RAM, or I/O
  - Removal of unnecessary logic
  - Changes to synthesis directives
2. Click the Rapid Recompile Preservation Summary report to view detailed information about the percentage of preserved compilation results.

**Figure 93. Rapid Recompile Preservation Summary**

| Rapid Recompile Preservation Summary |                         |                           |
|--------------------------------------|-------------------------|---------------------------|
|                                      | Type                    | Achieved                  |
| 1                                    | Placement (by node)     | 33.25 % ( 2160 / 6497 )   |
| 2                                    | Routing (by connection) | 49.93 % ( 14165 / 28372 ) |

## 6.7 Generating Programming Files

The Compiler's Assembler module generates device programming files. Run the Assembler to generate device programming files following successful design place and route.

1. Before running the Assembler, specify settings to customize programming file generation. Click **Assignments > Device > Device & Pin Options** to enable or disable generation of optional programming files.
2. To generate device programming files, click **Processing > Start > Start Assembler**, or click **Assembler** on the Compilation Dashboard. The Compiler confirms that prerequisite modules are complete, and launches the Assembler module to generate the programming files that you specify. The Messages window dynamically displays processing information, warnings, or errors. After Assembler processing,

After running the Assembler, the Compilation report provides detailed information about programming file generation, including programming file Summary and Encrypted IP information.

Figure 94. Assembler Reports

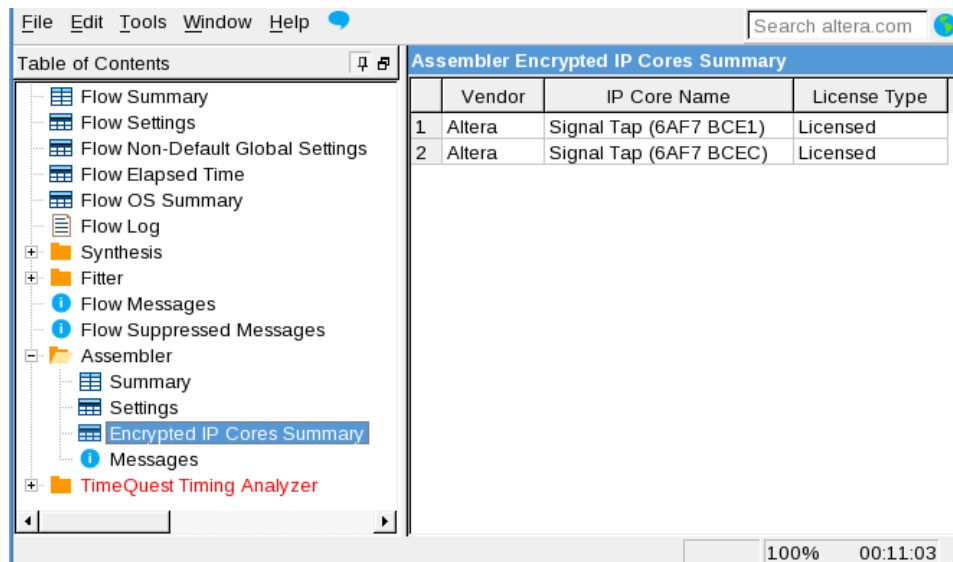
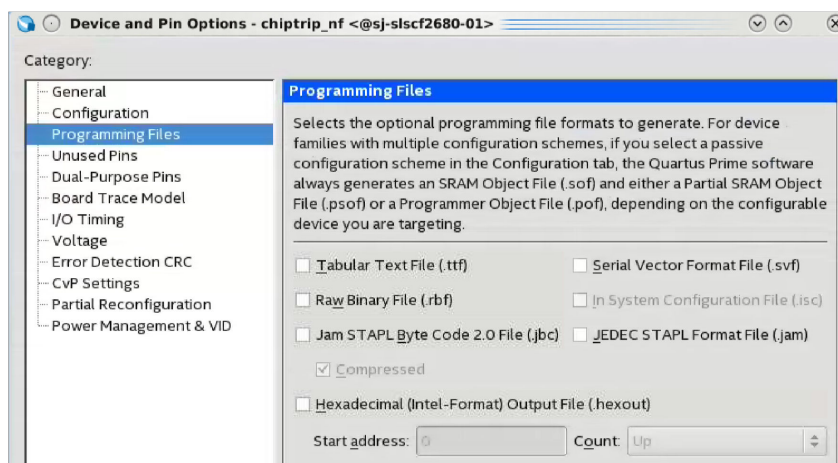


Figure 95. Device & Pin Options



**Related Links**

[Programming Intel FPGA Devices](#)

## 6.8 Synthesis Language Support

The Intel Quartus Prime software synthesizes standard Verilog HDL, VHDL, and SystemVerilog design files.

### 6.8.1 Verilog and SystemVerilog Synthesis Support

Intel Quartus Prime synthesis supports the following Verilog HDL language standards:

- Verilog-1995 (IEEE Standard 1364-1995)
- Verilog-2001 (IEEE Standard 1364-2001)
- SystemVerilog-2005 (IEEE Standard 1800-2005)
- SystemVerilog-2009 (IEEE Standard 1800-2009)

The following important guidelines apply to Intel Quartus Prime synthesis of Verilog HDL and SystemVerilog:

- The Compiler uses the Verilog-2001 standard by default for files with an extension of `.v`, and the SystemVerilog standard for files with the extension of `.sv`.
- If you use scripts to add design files, you can use the `-HDL_VERSION` command to specify the HDL version for each design file.
- Compiler support for Verilog HDL is case sensitive in accordance with the Verilog HDL standard.
- The Compiler supports the compiler directive ``define`, in accordance with the Verilog HDL standard.
- The Compiler supports the `include` compiler directive to include files with absolute paths (with either `"/"` or `"\"` as the separator), or relative paths.
- When searching for a relative path, the Compiler initially searches relative to the project directory. If the Compiler cannot find the file, the Compiler next searches relative to all user libraries. Finally, the Compiler searches relative to the current file's directory location.
- Intel Quartus Prime Pro Edition synthesis searches for all modules or entities earlier in the synthesis process than other Quartus software tools. This earlier search produces earlier syntax errors for undefined entities than other Quartus software tools.

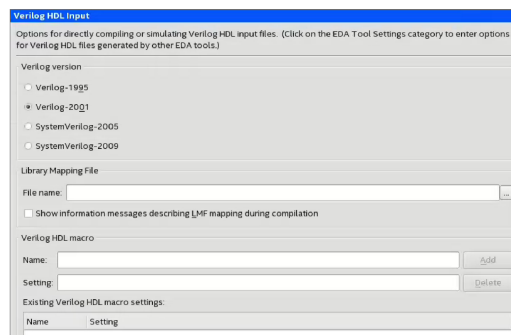
### Related Links

- [The Quartus Prime Timing Analyzer](#)
- [Recommended Design Practices](#)
- [Recommended HDL Coding Styles](#)

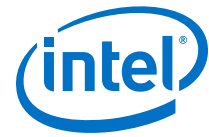
#### 6.8.1.1 Verilog HDL Input Settings (Settings Dialog Box)

Click **Assignments** ► **Settings** ► **Verilog HDL Input** to specify options for the synthesis of Verilog HDL input files.

**Figure 96. Verilog HDL Input Settings Dialog Box**







**Table 45. Verilog HDL Input Settings**

| Setting                     | Description                                                                                                                                                                                                                                                 |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Verilog Version</b>      | Directs synthesis to process Verilog HDL input design files using the specified standard. You can select any of the supported language standards to match your Verilog HDL files or SystemVerilog design files.                                             |
| <b>Library Mapping File</b> | Allows you to optionally specify a provided Library Mapping File (.lmf) for use in synthesizing Verilog HDL files that contain non-Intel FPGA functions mapped to IP cores. You can specify the full path name of the LMF in the <b>File name</b> box.      |
| <b>Verilog HDL Macro</b>    | Verilog HDL macros are pre-compiler directives which can be added to Verilog HDL files to define constants, flags, or other features by <b>Name</b> and <b>Setting</b> . Macros that you add appear in the <b>Existing Verilog HDL macro settings</b> list. |

### 6.8.1.2 Design Libraries

By default, the Compiler processes all design files into one or more libraries.

- When compiling a design instance, the Compiler initially searches for the entity in the library associated with the instance (which is the work library if you do not specify any library).
- If the Compiler cannot locate the entity definition, the Compiler searches for a unique entity definition in all design libraries.
- If the Compiler finds more than one entity with the same name, the Compiler generates an error. If your design uses multiple entities with the same name, you must compile the entities into separate libraries.

### 6.8.1.3 Verilog HDL Configuration

Verilog HDL configuration is a set of rules that specify the source code for particular instances. Verilog HDL configuration allows you to perform the following tasks:

- Specify a library search order for resolving cell instances (as does a library mapping file).
- Specify overrides to the logical library search order for specified instances.
- Specify overrides to the logical library search order for all instances of specified cells.

#### Related Links

[Configuration Syntax](#)

#### 6.8.1.3.1 Hierarchical Design Configurations

A design can have more than one configuration. For example, you can define a configuration that specifies the source code you use in particular instances in a sub-hierarchy, and then define a configuration for a higher level of the design.

For example, suppose a subhierarchy of a design is an eight-bit adder, and the RTL Verilog code describes the adder in a logical library named `rtlLib`. The gate-level code describes the adder in the `gateLib` logical library. If you want to use the gate-level code for the 0 (zero) bit of the adder and the RTL level code for the other seven bits, the configuration might appear as follows:

#### Example 63. Gate-level code for the 0 (zero) bit of the adder

```
config cfg1;
design aLib.eight_adder;
default liblist rtlLib;
instance adder.fulladd0 liblist gateLib;
endconfig
```

If you are instantiating this eight-bit adder eight times to create a 64-bit adder, use configuration `cfg1` for the first instance of the eight-bit adder, but not in any other instance. A configuration that performs this function is shown below:

#### Example 64. Use configuration `cfg1` for first instance of eight-bit adder

```
config cfg2;
design bLib.64_adder;
default liblist bLib;
instance top.64add0 use work.cfg1:config;
endconfig
```

**Note:** The name of the unbound module may be different from the name of the cell that is bounded to the instance.

### 6.8.1.4 Initial Constructs and Memory System Tasks

The Intel Quartus Prime software infers power-up conditions from the Verilog HDL `initial` constructs. The Intel Quartus Prime software also creates power-up settings for variables, including RAM blocks. If the Intel Quartus Prime software encounters non-synthesizable constructs in an `initial` block, it generates an error.

To avoid such errors, enclose non-synthesizable constructs (such as those intended only for simulation) in `translate_off` and `translate_on` synthesis directives. Synthesis of initial constructs enables the power-up state of the synthesized design to match the power-up state of the original HDL code in simulation.

**Note:** Initial blocks do not infer power-up conditions in some third-party EDA synthesis tools. If you convert between synthesis tools, you must set your power-up conditions correctly.

Intel Quartus Prime synthesis supports the `$readmemb` and `$readmemh` system tasks to initialize memories.

#### Example 65. Verilog HDL Code: Initializing RAM with the `readmemb` Command

```
reg [7:0] ram[0:15];
initial
begin
  $readmemb("ram.txt", ram);
end
```



When creating a text file to use for memory initialization, specify the address using the format @<location> on a new line, and then specify the memory word such as 110101 or abcde on the next line.

The following example shows a portion of a Memory Initialization File (.mif) for the RAM.

#### Example 66. Text File Format: Initializing RAM with the readmemb Command

```
@0  
00000000  
@1  
00000001  
@2  
00000010  
...  
@e  
00001110  
@f  
00001111
```

#### Related Links

- [Translate Off and On / Synthesis Off and On](#)
- [Incremental Compilation for Hierarchical and Team-Based Design](#)

### 6.8.1.5 Verilog HDL Macros

The Intel Quartus Prime software fully supports Verilog HDL macros, which you can define with the `'define` compiler directive in your source code. You can also define macros in the Intel Quartus Prime software or on the command line.

To set Verilog HDL macros at the command line for the Intel Quartus Prime Pro Edition synthesis (`quartus_syn`) executable, use the following format:

```
quartus_syn <PROJECT_NAME> --set=VERILOG_MACRO=a=2
```

This command adds the following new line to the project .qsf file:

```
set_global_assignment -name VERILOG_MACRO "a=2"
```

To avoid adding this line to the project .qsf, add this option to the `quartus_syn` command:

```
--write_settings_files=off
```

### 6.8.2 VHDL Synthesis Support

Intel Quartus Prime synthesis supports the following VHDL language standards.

- VHDL 1987 (IEEE Standard 1076-1987)
- VHDL 1993 (IEEE Standard 1076-1993)
- VHDL 2008 (IEEE Standard 1076-2008)

The Intel Quartus Prime Compiler uses the VHDL 1993 standard by default for files that have the extension `.vhdl` or `.vhd`.

**Note:** The VHDL code samples follow the VHDL 1993 standard.

## Related Links

[Migrating to Quartus Prime Pro Edition](#)

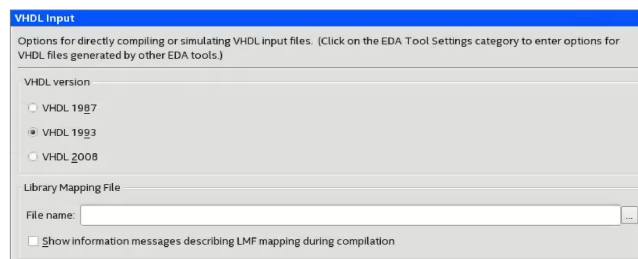
### 6.8.2.1 VHDL Input Settings (Settings Dialog Box)

Click **Assignments** > **Settings** > **VHDL Input** to specify options for the synthesis of VHDL input files.

**Table 46. VHDL Input Settings**

| Setting                     | Description                                                                                                                                                          |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>VHDL Version</b>         | Specifies the VHDL standard for use during synthesis of VHDL input design files. Select the language standards that corresponds with the VHDL files.                 |
| <b>Library Mapping File</b> | Specifies a Library Mapping File (.lmf) for use in synthesizing VHDL files that contain IP cores. Specify the full path name of the LMF in the <b>File name</b> box. |

**Figure 97. VHDL Input Settings Dialog Box**



### 6.8.2.2 VHDL Standard Libraries and Packages

The Intel Quartus Prime software includes the standard IEEE libraries and several vendor-specific VHDL libraries. The IEEE library includes the standard VHDL packages `std_logic_1164`, `numeric_std`, `numeric_bit`, and `math_real`.

The STD library is part of the VHDL language standard and includes the packages `standard` (included in every project by default) and `textio`. For compatibility with older designs, the Intel Quartus Prime software also supports the following vendor-specific packages and libraries:

- Synopsys packages such as `std_logic_arith` and `std_logic_unsigned` in the IEEE library.
- Mentor Graphics\* packages such as `std_logic_arith` in the ARITHMETIC library.
- Primitive packages `altera_primitives_components` (for primitives such as GLOBAL and DFFE) and `maxplus2` in the ALTERA library.
- IP core packages `altera_mf_components` in the ALTERA\_MF library for specific IP cores including LCELL. In addition, `lpm_components` in the LPM library for library of parameterized modules (LPM) functions.

**Note:** Import component declarations for primitives such as GLOBAL and DFFE from the `altera_primitives_components` package and not the `altera_mf_components` package.



### 6.8.2.3 VHDL wait Constructs

The Intel Quartus Prime software supports one VHDL `wait until` statement per process block. However, the Intel Quartus Prime software does not support other VHDL wait constructs, such as `wait for` and `wait on` statements, or processes with multiple wait statements.

#### Example 67. VHDL `wait until` construct example

```
architecture dff_arch of ls_dff is
begin
output: process begin
wait until (CLK'event and CLK='1');
Q <= D;
Qbar <= not D;
end process output;
end dff_arch;
```

## 6.9 Synthesis Settings Reference

This section provides a reference to all synthesis settings. Use these settings to customize synthesis processing for your design goals.

### 6.9.1 Optimization Modes

The following options direct the focus of Compiler optimization efforts during synthesis. The settings affect synthesis and fitting.

Table 47. Optimization Modes (Compiler Settings Page)

| Optimization Mode                                                  | Description                                                                                                                                  |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Balanced (Normal Flow)</b>                                      | Optimizes synthesis for balanced implementation that respects timing constraints.                                                            |
| <b>Performance (High effort - increases runtime)</b>               | Makes high effort to optimize synthesis for speed performance. High effort increases synthesis run time.                                     |
| <b>Performance (Aggressive - increases runtime and area)</b>       | Makes aggressive effort to optimize synthesis for speed performance. Aggressive effort increases synthesis run time and device resource use. |
| <b>Power (High effort - increases runtime)</b>                     | Makes high effort to optimize synthesis for low power. High effort increases synthesis run time.                                             |
| <b>Power (Aggressive - increases runtime, reduces performance)</b> | Makes aggressive effort to optimize synthesis for low power. Aggressive effort increases synthesis time and reduces speed performance.       |
| <b>Area (Aggressive - reduces performance)</b>                     | Makes aggressive effort to reduce the device area required to implement the design.                                                          |

### 6.9.2 Prevent Register Retiming

The **Prevent Register Retiming** option controls whether or not to globally disable retiming. When set to **disabled**, the Compiler automatically performs register retiming optimizations, moving registers through combinational logic. When set to **enabled**, the Compiler prevents any retiming optimizations on a global scale. Optionally, assign **Allow Register Retiming** to any design entity or instance to override **Prevent Register Retiming** for specific portions of the design. Click **Assignments > Assignment Editor** to specify entity- and instance-level assignments, or use the following syntax to make the assignment in the `.qsf` directly.

### Example 68. Disable register retiming for entity abc

```
set_global_assignment -name ALLOW_REGISTER_RETIMING ON
set_instance_assignment -name ALLOW_REGISTER_RETIMING OFF -to "abc|"
set_instance_assignment -name ALLOW_REGISTER_RETIMING ON -to "abc|def|"
```

### Example 69. Disable register retiming for the whole design, except for registers in entity abc

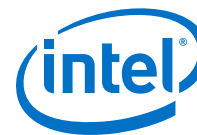
```
set_global_assignment -name ALLOW_REGISTER_RETIMING OFF
set_instance_assignment -name ALLOW_REGISTER_RETIMING ON -to "abc|"
set_instance_assignment -name ALLOW_REGISTER_RETIMING OFF -to "abc|def|"
```

## 6.9.3 Advanced Synthesis Settings

The following section is a quick reference of all Advanced Synthesis Settings. Click **Assignments > Settings > Compiler Settings > Advanced Settings (Synthesis)** to modify these settings.

**Table 48. Advanced Synthesis Settings (1 of 13)**

| Option                                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Allow Any RAM Size for Recognition</b>              | Allows the Compiler to infer RAMs of any size, even if the RAMs do not meet the current minimum requirements.                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Allow Any ROM Size for Recognition</b>              | Allows the Compiler to infer ROMs of any size even if the ROMs do not meet the design's current minimum size requirements.                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Allow Any Shift Register Size for Recognition</b>   | Allows the Compiler to infer shift registers of any size even if they do not meet the design's current minimum size requirements.                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Allow Register Duplication</b>                      | Controls whether the Compiler duplicates registers to improve design performance. When enabled, the Compiler performs optimization that creates a second copy of a register and move a portion of its fan-out to this new node. This technique improves routability and reduces the total routing wire required to route a net with many fan-outs. If you disable this option, retiming of registers is also disabled.<br><i>Note:</i> Not available for Intel Stratix 10 devices.                        |
| <b>Allow Register Merging</b>                          | Controls whether the Compiler removes (merges) identical registers. When enabled, in cases where two registers generate the same logic, the Compiler may delete one register and fan-out the remaining register to the deleted register's destinations. This option is useful if you want to prevent the Compiler from removing duplicate registers that you have used deliberately. When disabled, retiming optimizations are also disabled.<br><i>Note:</i> Not available for Intel Stratix 10 devices. |
| <b>Allow Shift Register Merging Across Hierarchies</b> | Allows the Compiler to take shift registers from different hierarchies of the design and put the registers in the same RAM.                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Allow Synchronous Control Signals</b>               | Allows the Compiler to utilize synchronous clear and synchronous load signals in normal mode logic cells. Turning on this option helps to reduce the total number of logic cells used in the design, but can negatively impact the fitting. This negative impact occurs because all the logic cells in a LAB share synchronous control signals.                                                                                                                                                           |



**Table 49. Advanced Synthesis Settings (2 of 13)**

| Option                                        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Analysis &amp; Synthesis Message Level</b> | Specifies the type of Analysis & Synthesis messages the Compiler display. <b>Low</b> displays only the most important Analysis & Synthesis messages. <b>Medium</b> displays most messages, but hides the detailed messages. <b>High</b> displays all messages.                                                                                                                                                                                                                                                   |
| <b>Auto Carry Chains</b>                      | Allows the Compiler to create carry chains automatically by inserting CARRY_SUM buffers into the design. The <b>Carry Chain Length</b> option controls the length of the chains. When this option is off, the Compiler ignores CARRY buffers, but CARRY_SUM buffers are unaffected. The Compiler ignores the <b>Auto Carry Chains</b> option if you select <b>Product Term</b> or <b>ROM</b> as the setting for the <b>Technology Mapper</b> option.<br><i>Note:</i> Not available for Intel Stratix 10 devices. |
| <b>Auto Clock Enable Replacement</b>          | Allows the Compiler to locate logic that feeds a register and move the logic to the register's clock enable input port.                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Auto DSP Block Replacement</b>             | Allows the Compiler to find a multiply-accumulate function or a multiply-add function that can be replaced with a DSP block.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Auto Gated Clock Conversion</b>            | Automatically converts gated clocks to use clock enable pins. Clock gating logic can contain AND, OR, MUX, and NOT gates. Turning on this option may increase memory use and overall run time. You must use the Timing Analyzer for timing analysis, and you must define all base clocks in Synopsys Design Constraints (.sdc) format.                                                                                                                                                                           |

**Table 50. Advanced Synthesis Settings (3 of 13)**

| Option                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Auto Open-Drain Pins</b>          | Allows the Compiler to automatically convert a tri-state buffer with a strong low data input into the equivalent open-drain buffer.                                                                                                                                                                                                                                                                                                                                   |
| <b>Auto RAM Replacement</b>          | Allows the Compiler to identify sets of registers and logic that it can replace with the altsyncram or the lpm_ram_dp IP core. Turning on this option may change the functionality of the design.                                                                                                                                                                                                                                                                     |
| <b>Auto ROM Replacement</b>          | Allows the Compiler to identify logic that it can replace with the altsyncram or the lpm_rom IP core. Turning on this option may change the power-up state of the design.                                                                                                                                                                                                                                                                                             |
| <b>Auto Resource Sharing</b>         | Allows the Compiler to share hardware resources among many similar, but mutually exclusive, operations in your HDL source code. If you enable this option, the Compiler merges compatible addition, subtraction, and multiplication operations. Merging operations may reduce the area your design requires. Because resource sharing introduces extra muxing and control logic on each shared resource, it may negatively impact the final $f_{MAX}$ of your design. |
| <b>Auto Shift Register Placement</b> | Allows the Compiler to find a group of shift registers of the same length that are replaceable with the altshift_taps IP core. The shift registers must all use the same clock and clock enable signals. The registers must not have any other secondary signals. The registers must have equally spaced taps that are at least three registers apart.                                                                                                                |
| <b>Automatic Parallel Synthesis</b>  | Option to enable/disable automatic parallel synthesis. Use this option to speed up synthesis compile time by using multiple processors when available.                                                                                                                                                                                                                                                                                                                |

**Table 51. Advanced Synthesis Settings (4 of 13)**

| Option                     | Description                                                                                                                                                                                                                                                               |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Block Design Naming</b> | Specifies the naming scheme for the block design. The Compiler ignores the option if you assign the option to anything other than a design entity.                                                                                                                        |
| <b>Carry Chain Length</b>  | Specifies the maximum allowable length of a chain for CARRY_SUM buffers, including those that you or the Compiler instantiate. The Compiler breaks carry chains that exceed this length into separate chains.<br><i>Note:</i> Not available for Intel Stratix 10 devices. |
| <i>continued...</i>        |                                                                                                                                                                                                                                                                           |

| Option                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Clock MUX Protection</b>                | Causes the multiplexers in the clock network to decompose to 2-to-1 multiplexer trees. The Compiler protects these trees from merging with, or transferring to, other logic. This option helps the Timing Analyzer to analyze clock behavior.                                                                                                                                                                                                                                                                                 |
| <b>Create Debugging Nodes for IP Cores</b> | Makes certain nodes (for example, important registers, pins, and state machines) visible for all the IP cores in a design. Use IP core nodes to effectively debug the IP core. This technique is effective when using the IP core with the Signal Tap Logic Analyzer. The Node Finder, using Signal Tap Logic Analyzer filters, displays all the nodes that Analysis & Synthesis makes visible. When making the debugging nodes visible, Analysis & Synthesis can change the $f_{MAX}$ and number of logic cells in IP cores. |
| <b>DSP Block Balancing</b>                 | Allows you to control the conversion of certain DSP block slices during DSP block balancing.                                                                                                                                                                                                                                                                                                                                                                                                                                  |

**Table 52. Advanced Synthesis Settings (5 of 13)**

| Option                                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Disable DSP Negate Inferencing</b>              | Allows you to specify whether to use the negate port on an inferred DSP block.                                                                                                                                                                                                                                                                                                                                                               |
| <b>Disable Register Merging Across Hierarchies</b> | Specifies whether the Compiler allows merging of registers that are in different hierarchies if their inputs are the same.                                                                                                                                                                                                                                                                                                                   |
| <b>Enable State Machines Inference</b>             | Allows the Compiler to infer state machines from VHDL or Verilog HDL design files. The Compiler optimizes state machines to reduce area and improve performance. If set to <b>Off</b> , the Compiler extracts and optimizes state machines in VHDL or Verilog HDL design files as regular logic.                                                                                                                                             |
| <b>Force Use of Synchronous Clear Signals</b>      | Forces the Compiler to utilize synchronous clear signals in normal mode logic cells. Enabling this option helps to reduce the total number of logic cells in the design, but can negatively impact the fitting. All the logic cells in a LAB share synchronous control signals.                                                                                                                                                              |
| <b>HDL Message Level</b>                           | Specifies the type of HDL messages you want to view, including messages that display processing errors in the HDL source code. <b>Level1</b> displays only the most important HDL messages. <b>Level2</b> displays most HDL messages, including warning and information based messages. <b>Level3</b> displays all HDL messages, including warning and information based messages and alerts about potential design problems or lint errors. |
| <b>Ignore CARRY Buffers</b>                        | Ignores CARRY_SUM buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual CARRY_SUM buffer, or to a design entity containing CARRY_SUM buffers.                                                                                                                                                                                                                                 |
| <b>Ignore CASCADE Buffers</b>                      | Ignores CASCADE buffers that are instantiated in the design. The Compiler ignores this option if you apply the option to anything other than an individual CASCADE buffer, or a design entity containing CASCADE buffers.                                                                                                                                                                                                                    |

**Table 53. Advanced Synthesis Settings (6 of 13)**

| Option                       | Description                                                                                                                                                                                      |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Ignore GLOBAL Buffers</b> | Ignores GLOBAL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual GLOBAL buffer, or a design entity containing GLOBAL buffers. |
| <b>Ignore LCELL Buffers</b>  | Ignores LCELL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual LCELL buffer, or a design entity containing LCELL buffers.    |
| <i>continued...</i>          |                                                                                                                                                                                                  |





| Option                                    | Description                                                                                                                                                                                         |
|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Ignore Maximum Fan-Out Assignments</b> | Directs the Compiler to ignore the Maximum Fan-Out Assignments on a node, an entity, or the whole design.                                                                                           |
| <b>Ignore ROW GLOBAL Buffers</b>          | Ignores ROW GLOBAL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual GLOBAL buffer or a design entity containing GLOBAL buffers. |
| <b>Ignore SOFT Buffers</b>                | Ignores SOFT buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual SOFT buffer or a design entity containing SOFT buffers.           |

**Table 54. Advanced Synthesis Settings (7 of 13)**

| Option                                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Ignore translate_off and synthesis_off Directives</b> | Ignores all translate_off/synthesis_off synthesis directives in Verilog HDL and VHDL design files. Use this option to disable these synthesis directives and include previously ignored code during elaboration.                                                                                                                                                                                                   |
| <b>Infer RAMs from Raw Logic</b>                         | Infers RAM from registers and multiplexers. The Compiler initially converts some HDL patterns differing from RAM templates into logic. However, these structures function as RAM. As a result, when you enable this option, the Compiler may substitute the altsyncram IP core instance for them at a later stage. When you enable this assignment, the Compiler may use more device RAM resources and fewer LABs. |
| <b>Iteration Limit for Constant Verilog Loops</b>        | Defines the iteration limit for Verilog loops with loop conditions that evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.                                                                                                                                    |
| <b>Iteration Limit for non-Constant Verilog Loops</b>    | Defines the iteration limit for Verilog HDL loops with loop conditions that do not evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.                                                                                                                         |

**Table 55. Advanced Synthesis Settings (8 of 13)**

| Option                                                   | Description                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Limit AHDL integers to 32 Bits</b>                    | Specifies whether an AHDL-based design must have a limit on integer size of 32 bits. The Compiler provides this option for backward compatibility with pre-2000.09 releases of the Intel Quartus Prime software. Such registers do not support integers larger than 32 bits in AHDL.                                                                                                                |
| <b>Maximum DSP Block Usage</b>                           | Specifies the maximum number of DSP blocks that the DSP block balancer assumes exist in the current device for each partition. This option overrides the usual method of using the maximum number of DSP blocks the current device supports.                                                                                                                                                        |
| <b>Maximum Number of LABs</b>                            | Specifies the maximum number of LABs that Analysis & Synthesis should try to utilize for a device. This option overrides the usual method of using the maximum number of LABs the current device supports, when the value is non-negative and is less than the maximum number of LABs available on the current device.                                                                              |
| <b>Maximum Number of M4K/M9K/M20K/M10K Memory Blocks</b> | Specifies the maximum number of M4K, M9K, M20K, or M10K memory blocks that the Compiler may use for a device. This option overrides the usual method of using the maximum number of M4K, M9K, M20K, or M10K memory blocks the current device supports, when the value is non-negative and is less than the maximum number of M4K, M9K, M20K, or M10K memory blocks available on the current device. |

**Table 56. Advanced Synthesis Settings (9 of 13)**

| Option                                                          | Description                                                                                                                                                                                                                                                                                                                                      |
|-----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Maximum Number of Registers Created from Uninferred RAMs</b> | Specifies the maximum number of registers that Analysis & Synthesis uses for conversion of uninferred RAMs. Use this option as a project-wide option or on a specific partition by setting the assignment on the instance name of the partition root. The assignment on a partition overrides the global assignment (if any) for that particular |

*continued...*

| Option                                                                    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                           | partition. This option prevents synthesis from causing long compilations and running out of memory when many registers are used for uninferred RAMs. Instead of continuing the compilation, the Intel Quartus Prime software issues an error and exits.                                                                                                                                                                                                                                                                                                                                                         |
| <b>NOT Gate Push-Back</b>                                                 | Allows the Compiler to push an inversion (that is, a NOT gate) back through a register and implement it on that register's data input if it is necessary to implement the design. When this option is on, a register may power-up to an active-high state, and may need explicit clear during initial operation of the device. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers. When you apply this option to an output pin that is directly fed by a register, the assignment automatically transfers to that register. |
| <b>Number of Inverted Registers Reported in Synthesis Report</b>          | Specifies the maximum number of inverted registers that the Synthesis report displays.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Number of Protected Registers Reported in Synthesis Report</b>         | Specifies the maximum number of protected registers that the Synthesis Report displays.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Number of Removed Registers Reported in Synthesis Migration Checks</b> | Specifies the maximum number of rows that the Synthesis Migration Check report displays.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Number of Swept Nodes Reported in Synthesis Report</b>                 | Specifies the maximum number of swept nodes that the Synthesis Report displays. A swept node is any node which was eliminated from your design because the Compiler found the node to be unnecessary.                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Number of Rows Reported in Synthesis Report</b>                        | Specifies the maximum number of rows that the Synthesis report displays.<br><i>Note:</i> Not available for Intel Stratix 10 devices.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Optimization Technique</b>                                             | Specifies an overall optimization goal for Analysis & Synthesis. The Compiler can maximize synthesis processing for performance, minimize logic usage, or balance high performance with minimal logic usage.                                                                                                                                                                                                                                                                                                                                                                                                    |

**Table 57. Advanced Synthesis Settings (10 of 13)**

| Option                                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Perform WYSIWYG Primitive Resynthesis</b> | Specifies whether to perform WYSIWYG primitive resynthesis during synthesis. This option uses the setting specified in the <b>Optimization Technique</b> logic option.                                                                                                                                                                                                                                                                                                                |
| <b>Power-Up Don't Care</b>                   | Causes registers that do not have a <b>Power-Up Level logic</b> option setting to power-up with a do not care logic level (X). When the <b>Power-Up Don't Care</b> option is on, the Compiler determines when it is beneficial to change the power-up level of a register to minimize the area of the design. The Compiler maintains a power-up state of zero, unless there is an immediate area advantage.                                                                           |
| <b>Power Optimization During Synthesis</b>   | Controls the power-driven compilation setting of Analysis & Synthesis. This option determines how aggressively Analysis & Synthesis optimizes the design for power. When this option is <b>Off</b> , the Compiler does not perform any power optimizations. <b>Normal compilation</b> performs power optimizations provided that they are not expected to reduce design performance. <b>Extra effort</b> performs additional power optimizations which may reduce design performance. |

**Table 58. Advanced Synthesis Settings (11 of 13)**

| Option                            | Description                                                                                                                                                                                                                                                                                                                                            |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Remove Duplicate Registers</b> | Removes a register if it is identical to another register. If two registers generate the same logic, the Compiler deletes the duplicate. The first instance fans-out to the duplicates destinations. Also, if the deleted register contains different logic option assignments, the Compiler ignores the options. This option is useful if you want to |

*continued...*



| Option                              | Description                                                                                                                                                                                                       |
|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                     | prevent the Compiler from removing intentionally duplicate registers. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers.     |
| <b>Remove Redundant Logic Cells</b> | Removes redundant LCELL primitives or WYSIWYG primitives. Turning this option on optimizes a circuit for area and speed. The Compiler ignores this option if you apply it to anything other than a design entity. |
| <b>Report Connectivity Checks</b>   | Specifies whether the Synthesis report includes reports under the <b>Connectivity Checks</b> folder.<br><i>Note:</i> Not available for Intel Stratix 10 devices.                                                  |
| <b>Report Parameter Settings</b>    | Specifies whether the Synthesis report includes the reports in the <b>Parameter Settings by Entity Instance</b> folder.                                                                                           |
| <b>Report Source Assignments</b>    | Specifies whether the Synthesis report includes reports in the <b>Source Assignments</b> folder.                                                                                                                  |

**Table 59. Advanced Synthesis Settings (12 of 13)**

| Option                                                              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Resource Aware Inference for Block RAM</b>                       | Specifies whether RAM, ROM, and shift-register inference should take the design and device resources into account.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Restructure Multiplexers</b>                                     | Reduces the number of logic elements synthesis requires to implement multiplexers in a design. This option is useful if your design contains buses of fragmented multiplexers. This option repacks multiplexers more efficiently for area, allowing the design to implement multiplexers with a reduced number of logic elements: <ul style="list-style-type: none"> <li>• <b>On</b>—minimizes your design area, but may negatively affect design clock speed (<math>f_{MAX}</math>).</li> <li>• <b>Off</b>—disables multiplexer restructuring; it does not decrease logic element usage and does not affect design clock speed (<math>f_{MAX}</math>).</li> <li>• <b>Auto</b>—allows the Intel Quartus Prime software to determine whether multiplexer restructuring should be enabled. The <b>Auto</b> setting decreases logic element usage, but may negatively affect design clock speed (<math>f_{MAX}</math>).</li> </ul> |
| <b>SDC Constraint Protection</b>                                    | Verifies .sdc constraints in register merging. This option helps to maintain the validity of .sdc constraints through compilation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Safe State Machine</b>                                           | Directs the Compiler to implement state machines that can recover from an illegal state.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Shift Register Replacement – Allow Asynchronous Clear Signal</b> | Allows the Compiler to find a group of shift registers of the same length that can be replaced with the altshift_taps IP core. The shift registers must all use the same aclr signals, must not have any other secondary signals, and must have equally spaced taps that are at least three registers apart. To use this option, you must turn on the <b>Auto Shift Register Replacement</b> logic option.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

**Table 60. Advanced Synthesis Settings (13 of 13)**

| Option                                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>State Machine Processing</b>              | Specifies the processing style the Compiler uses to process a state machine. You can use your own <b>User-Encoded</b> style, or select <b>One-Hot</b> , <b>Minimal Bits</b> , <b>Gray</b> , <b>Johnson</b> , <b>Sequential</b> , or <b>Auto</b> (Compiler-selected) encoding.                                                                                                                                                                                                                                                                                    |
| <b>Strict RAM Replacement</b>                | When this option is <b>On</b> , the Compiler replace RAM only if the hardware matches the design exactly.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Synchronization Register Chain Length</b> | Specifies the maximum number of registers in a row that the Compiler considers as a synchronization chain. Synchronization chains are sequences of registers with the same clock and no fan-out in between, such that the first register is fed by a pin, or by logic in another clock domain. The Compiler considers these registers for metastability analysis. The Compiler prevents optimizations of these registers, such as retiming. When gate-level retiming is enabled, the Compiler does not remove these registers. The default length is set to two. |
| <i>continued...</i>                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

| Option                                          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synthesis Effort</b>                         | Controls the synthesis trade-off between compilation speed, performance, and area. The default is <b>Auto</b> . You can select <b>Fast</b> for faster compilation speed at the cost of performance and area.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Synthesis Migration Check for Stratix 10</b> | Enables synthesis checks on Intel Arria 10 to Intel Stratix 10 design migration.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Timing-Driven Synthesis</b>                  | Allows synthesis to use timing information to better optimize the design. The <b>Timing-Driven Synthesis</b> logic option impacts the following <b>Optimization Technique</b> options: <ul style="list-style-type: none"> <li>• <b>Optimization Technique Speed</b>—optimizes timing-critical portions of your design for performance at the cost of increasing area (logic and register utilization)</li> <li>• <b>Optimization Technique Balanced</b>—also optimizes the timing-critical portions of your design for performance, but the option allows only limited area increase</li> <li>• <b>Optimization Technique Area</b>—optimizes your design only for area</li> </ul> |

## 6.10 Fitter Settings Reference

Use Fitter settings to customize the place and route of your design. Click **Assignments > Settings > Compiler Settings > Advanced Settings (Fitter)** to access Fitter settings.

**Table 61. Advanced Fitter Settings (1 of 8)**

| Option                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ALM Register Packing Effort</b> | Guides aggressiveness of the Fitter in packing ALMs during register placement. Use this option to increase secondary register locations. Increasing ALM packing density may lower the number of ALMs needed to fit the design, but it may also reduce routing flexibility and timing performance. <ul style="list-style-type: none"> <li>• <b>Low</b>—the Fitter avoids ALM packing configurations that combine LUTs and registers which have no direct connectivity. Avoiding these configurations may improve timing performance but increases the number of ALMs to implement the design.</li> <li>• <b>Medium</b>—the Fitter allows some configurations that combine unconnected LUTs and registers to be implemented in ALM locations. The Fitter makes more use of secondary register locations within the ALM.</li> <li>• <b>High</b>—the Fitter enables all legal and desired ALM packing configurations. In dense designs, the Fitter automatically increases the ALM register packing effort as required to enable the design to fit.</li> </ul> |
| <b>Allow Register Duplication</b>  | Allows the Compiler to duplicate registers to improve design performance. When you enable this option, the Compiler copies registers and moves some fan-out to this new node. This optimization improves routability and can reduce the total routing wire in nets with many fan-outs. If you disable this option, this disables optimizations that retime registers.<br><i>Note:</i> Not available for Intel Stratix 10 devices.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Allow Register Merging</b>      | Allows the Compiler to remove registers that are identical to other registers in the design. When you enable this option, in cases where two registers generate the same logic, the Compiler deletes one register, and the remaining registers fan-out to the deleted register's destinations. This option is useful if you want to prevent the Compiler from removing intentional use of duplicate registers.<br>If you disable register merging, the Compiler disables optimizations that retime registers.<br><i>Note:</i> Not available for Intel Stratix 10 devices.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>continued...</i>                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |



| Option                                              | Description                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Allow Delay Chains</b>                           | Allows the Fitter to choose the optimal delay chain to meet $t_{SU}$ and $t_{CO}$ timing requirements for all I/O elements. Enabling this option may reduce the number of $t_{SU}$ violations, while introducing a minimal number of $t_H$ violations. Enabling this option does not override delay chain settings on individual nodes.   |
| <b>Auto Delay Chains for High Fanout Input Pins</b> | Allows the Fitter to choose how to optimize the delay chains for high fan-out input pins. You must enable <b>Auto Delay Chains</b> to enable this option. Enabling this option may reduce the number of $t_{SU}$ violations, but the compile time increases significantly, as the Fitter tries to optimize the settings for all fan-outs. |
| <b>Auto Fit Effort Desired Slack Margin</b>         | Specifies the default worst-case slack margin the Fitter maintains for. If the design is likely to have at least this much slack on every path, the Fitter reduces optimization effort to reduce compilation time.                                                                                                                        |

**Table 62. Advanced Fitter Settings (2 of 8)**

| Option                                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Auto Global Clock</b>                    | Allows the Compiler to choose the global clock signal. The Compiler chooses the signal that feeds the most clock inputs to flip-flops. This signal is available throughout the device on the global routing paths. To prevent the Compiler from automatically selecting a particular signal as global clock, set the <b>Global Signal</b> option to <b>Off</b> on that signal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Auto Global Register Control Signals</b> | Allows the Compiler to choose global register control signals. The Compiler chooses signals that feed the most control signal inputs to flip-flops (excluding clock signals) as the global signals. These global signals are available throughout the device on the global routing paths. Depending on the target device family, these control signals can include asynchronous clear and load, synchronous clear and load, clock enable, and preset signals. If you want to prevent the Compiler from automatically selecting a particular signal as a global register control signal, set the <b>Global Signal</b> option to <b>Off</b> on that signal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Auto Packed Registers</b>                | <p>Allows the Compiler to combine a register and a combinational function, or to implement registers using I/O cells, RAM blocks, or DSP blocks instead of logic cells. This option controls how aggressively the Fitter combines registers with other function blocks to reduce the area of the design. Generally, the <b>Auto</b> or <b>Sparse Auto</b> settings are appropriate.</p> <p>The other settings limit the flexibility of the Fitter to combine registers with other function blocks and can result in no fits.</p> <ul style="list-style-type: none"> <li>• <b>Auto</b>—the Fitter attempts to achieve the best performance with good area. If necessary, the Fitter combines additional logic to reduce the area of the design to within the current device.</li> <li>• <b>Sparse Auto</b>—the Fitter attempts to achieve the highest performance, but may increase device usage without exceeding the device logic capacity.</li> <li>• <b>Off</b>—the Fitter does not combine registers with other functions. The <b>Off</b> setting severely increases the area of the design and may cause a no fit.</li> <li>• <b>Sparse</b>—the Fitter combines functions in a way which improves performance for many designs.</li> <li>• <b>Normal</b>—the Fitter combines functions that are expected to maximize design performance and reduce area.</li> <li>• <b>Minimize Area</b>—the Fitter aggressively combines unrelated functions to reduce the area required for placing the design, at the expense of performance.</li> <li>• <b>Minimize Area with Chains</b>—the Fitter even more aggressively combines functions that are part of register cascade chains or can be converted to register cascade chains.</li> </ul> |

*continued...*



| Option                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                    | If this option is set to any value but <b>Off</b> , registers combine with I/O cells to improve I/O timing. This remains true provided that the <b>Optimize IOC Register Placement For Timing</b> option is enabled.                                                                                                                                                                                                                                                                                                                             |
| <b>Auto RAM to MLAB Conversion</b> | Specifies whether the Fitter converts RAMs of <b>Auto</b> block type to use LAB locations. If this option is set to <b>Off</b> , only MLAB cells or RAM cells with a block type setting of <b>MLAB</b> use LAB locations to implement memory.                                                                                                                                                                                                                                                                                                    |
| <b>Auto Register Duplication</b>   | Allows the Fitter to automatically duplicate registers within a LAB that contains empty logic cells. This option does not alter the functionality of the design. The Compiler ignores the <b>Auto Register Duplication</b> option if you select <b>OFF</b> as the setting for the <b>Logic Cell Insertion -- Logic Duplication</b> logic option. Turning on this option allows the <b>Logic Cell Insertion -- Logic Duplication</b> logic option to improve a design's routability, but can make formal verification of a design more difficult. |

Table 63. Advanced Fitter Settings (3 of 8)

| Option                                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Enable Bus-Hold Circuitry</b>                        | Enables bus-hold circuitry during device operation. When this option is <b>On</b> , a pin retains its last logic level when it is not driven, and does not go to a high impedance logic level. Do not use this option at the same time as the <b>Weak Pull-Up Resistor</b> option. The Compiler ignores this option if you apply it to anything other than a pin.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Enable Critical Chain Viewer</b>                     | For Intel Stratix 10 designs, enables location to the Critical Chain Viewer from the Fast Forward Timing Closure Recommendations report. Use the Critical Chain Viewer to visualize the critical chains limiting further performance improvement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Equivalent RAM and MLAB Paused Read Capabilities</b> | Specifies whether RAMs implemented in MLAB cells must have equivalent paused read capabilities as RAMs implemented in block RAM. Pausing a read is the ability to keep around the last read value when reading is disabled. Allowing differences in paused read capabilities provides the Fitter more flexibility in implementing RAMs using MLAB cells. To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to <b>Don't Care</b> . The following options are available: <ul style="list-style-type: none"> <li>• <b>Don't Care</b>—the Fitter can convert RAMs to MLAB cells, even if they do not have equivalent paused read capabilities to a block RAM implementation. The Fitter generates an information message about RAMs with different paused read capabilities.</li> <li>• <b>Care</b>—the Fitter does not convert RAMs to MLAB cells unless they have the equivalent paused read capabilities to a block RAM implementation.</li> </ul>                                                                                  |
| <b>Equivalent RAM and MLAB Power Up</b>                 | Specifies whether RAMs implemented in MLAB cells must have equivalent power-up conditions as RAMs implemented in block RAM. Power-up conditions occur when the device powers-up or globally resets. Allowing non-equivalent power-up conditions provides the Fitter more flexibility in implementing RAMs using MLAB cells. To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to <b>Auto</b> or <b>Don't Care</b> . The following options are available: <ul style="list-style-type: none"> <li>• <b>Auto</b>—the Fitter may convert RAMs to MLAB cells, even if the MLAB cells lack equivalent power-up conditions to a block RAM implementation. The Fitter also outputs a warning message about RAMs with non-equivalent power up conditions.</li> <li>• <b>Don't Care</b>—the same behavior as <b>Auto</b> applies, but the message is an information message.</li> <li>• <b>Care</b>—the Fitter does not convert RAMs to MLAB cells unless they have equivalent power up conditions to a block RAM implementation.</li> </ul> |
| <b>Final Placement Optimizations</b>                    | Specifies whether the Fitter performs final placement optimizations. Performing final placement optimizations may improve timing and routability, but may also require longer compilation time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Fitter Aggressive Routability Optimizations</b>      | Specifies whether the Fitter aggressively optimizes for routability. Performing aggressive routability optimizations may decrease design speed, but may also reduce routing wire usage and routing time. The <b>Automatically</b> setting allows the Fitter to decide whether aggressive routability is beneficial.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |



**Table 64. Advanced Fitter Settings (4 of 8)**

| Option                                                                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Fitter Effort</b>                                                                   | <p>Specifies the level of physical synthesis optimization during fitting:</p> <ul style="list-style-type: none"> <li>• <b>Auto</b>—adjusts the Fitter optimization effort to minimize compilation time, while still achieving the design timing requirements. Use the <b>Auto Fit Effort Desired Slack Margin</b> option to apply sufficient optimization effort to achieve additional timing margin.</li> <li>• <b>Standard</b>—uses maximum effort regardless of the design's requirements, leading to higher compilation time and more margin on easier designs. For difficult designs, <b>Auto</b> and <b>Standard</b> both use maximum effort.</li> </ul>                                                                                                                                                                                                                                                                   |
| <b>Fitter Initial Placement Seed</b>                                                   | <p>Specifies the seed for the current design. The value can be any non-negative integer value. By default, the Fitter uses a seed of 1.</p> <p>The Fitter uses the seed as the initial placement configuration when optimizing design placement to meet timing requirements <math>f_{MAX}</math>. Because each different seed value results in a somewhat different fit, you can try several different seeds to attempt to obtain superior fitting results.</p> <p>The seeds that lead to the best fits for a design may change if the design changes. Also, changing the seed may or may not result in a better fit. Therefore, specify a seed only if the Fitter is not meeting timing requirements by a small amount.</p> <p><i>Note:</i> You can also use the Design Space Explorer II (DSEII) to sweep complex flow parameters, including the seed, in the Intel Quartus Prime software to optimize design performance.</p> |
| <b>Logic Cell Insertion</b>                                                            | <p>Allows the Fitter to automatically insert buffer logic cells between two nodes without altering the functionality of the design. The Compiler creates buffer logic cells from unused logic cells in the device. This option also allows the Fitter to duplicate a logic cell within a LAB when there are unused logic cells available in a LAB. Using this option can increase compilation time. The default setting of <b>Auto</b> allows these operations to run when the design requires them to fit the design.</p>                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>MLAB Add Timing Constraints for Mixed-Port Feed-Through Mode Setting Don't Care</b> | <p>Specifies whether the Timing Analyzer evaluates timing constraints between the write and the read operations of the MLAB memory block. Performing a write and read operation simultaneously at the same address might result in metastability issues because no timing constraints between those operations exist by default. Turning on this option introduces timing constraints between the write and read operations on the MLAB memory block and thereby avoids metastability issues. However, turning on this option degrades the performance of the MLAB memory blocks. If your design does not perform write and read operations simultaneously at the same address, you do not need to set this option.</p>                                                                                                                                                                                                          |

**Table 65. Advanced Fitter Settings (5 of 8)**

| Option                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Optimize Design for Metastability</b> | <p>This setting improves the reliability of the design by increasing its Mean Time Between Failures (MTBF). When you enable this setting, the Fitter increases the output setup slacks of synchronizer registers in the design. This slack can exponentially increase the design MTBF. This option only applies when using the Timing Analyzer for timing-driven compilation. Use the Timing Analyzer <code>report_metastability</code> command to review the synchronizers detected in your design and to produce MTBF estimates.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Optimize Hold Timing</b>              | <p>Directs the Fitter to optimize hold time within a device to meet timing requirements and assignments. The following settings are available:</p> <ul style="list-style-type: none"> <li>• <b>I/O Paths and Minimum TPD Paths</b>—directs the Fitter to meet the following timing requirements and assignments: <ul style="list-style-type: none"> <li>– <math>t_H</math> from I/O pins to registers.</li> <li>– Minimum <math>t_{CO}</math> from registers to I/O pins.</li> <li>– Minimum <math>t_{PD}</math> from I/O pins or registers to I/O pins or registers.</li> </ul> </li> <li>• <b>All Paths</b>—directs the Fitter to meet the following timing requirements and assignments: <ul style="list-style-type: none"> <li>– <math>t_H</math> from I/O pins to registers.</li> <li>– Minimum <math>t_{CO}</math> from registers to I/O pins.</li> <li>– Minimum <math>t_{PD}</math> from I/O pins or registers to I/O pins or registers.</li> </ul> </li> </ul> |

*continued...*

| Option                                            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                   | When you disable the <b>Optimize Timing</b> logic option, the <b>Optimize Hold Timing</b> option is not available.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Optimize IOC Register Placement for Timing</b> | Specifies whether the Fitter optimizes I/O pin timing by automatically packing registers into I/Os to minimize delays. <ul style="list-style-type: none"> <li>• <b>Normal</b>—the Fitter opportunistically packs registers into I/Os that should improve I/O timing.</li> <li>• <b>Pack All I/O Registers</b>— the Fitter aggressively packs any registers connected to input, output, or output enable pins into I/Os, unless prevented by your constraints or other legality restrictions.</li> <li>• <b>Off</b>—performs no periphery to core optimization.</li> </ul>                                                                                                                                                                                                                                                                 |
| <b>Optimize Multi-Corner Timing</b>               | Directs the Fitter to consider all timing corners during optimization to meet timing requirements. These timing delay corners include both fast-corner timing and slow-corner timing. By default, this option is <b>On</b> , and the Fitter optimizes designs considering multi-corner delays in addition to slow-corner delays. When this option is <b>Off</b> , the Fitter optimizes designs considering only slow-corner delays from the slow-corner timing model (slowest manufactured device for a given speed grade, operating in low-voltage conditions). Turning this option <b>On</b> typically creates a more robust design implementation across process, temperature, and voltage variations.<br>When you turn <b>Off</b> the <b>Optimize Timing</b> option, the <b>Optimize Multi-Corner Timing</b> option is not available. |
| <b>Optimize Timing</b>                            | Specifies whether the Fitter optimizes to meet the maximum delay timing requirements (for example, clock cycle time). By default, this option is set to <b>Normal compilation</b> . Turning this option <b>Off</b> helps fit designs that with extremely high interconnect requirements. Turning this option <b>Off</b> can also reduce compilation time at the expense of timing performance (because the Fitter ignores the design's timing requirements). If this option is <b>Off</b> , other Fitter timing optimization options have no effect (such as <b>Optimize Hold Timing</b> ).                                                                                                                                                                                                                                               |

Table 66. Advanced Fitter Settings (6 of 8)

| Option                                                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Optimize Timing for ECOs</b>                             | Controls whether the Fitter optimizes to meet your maximum delay timing requirements (for example, clock cycle time, $t_{SU}$ , $t_{CO}$ ) during ECO compiles. By default, this option is set to <b>Off</b> . Turning it <b>On</b> can improve timing performance at the cost of compilation time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Perform Clocking Topology Analysis During Routing</b>    | Directs the Fitter to perform an analysis of the design's clocking topology and adjust the optimization approach on paths with significant clock skew. Enabling this option may improve hold timing at the cost of increased compile time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Periphery to Core Placement and Routing Optimization</b> | Specifies whether the Fitter should perform targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core. The following options are available: <ul style="list-style-type: none"> <li>• <b>Auto</b>—the Fitter automatically identifies transfers with tight timing windows, places the core registers, and routes all connections to or from the periphery. The Fitter performs these placement and routing decisions before the rest of core placement and routing. This sequence ensures that these timing-critical connections meet timing, and also avoids routing congestion.</li> <li>• <b>On</b>— the Fitter optimizes all transfers between the periphery and core registers, regardless of timing requirements. Do not set this option to <b>On</b> globally. Instead, use the Assignment Editor to assign optimization to a targeted set of nodes or entities.</li> <li>• <b>Off</b>—the Fitter performs no periphery to core optimization.</li> </ul> |

*continued...*





| Option                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Physical Synthesis</b>                | Increases circuit performance by performing combinational and sequential optimization during fitting.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Placement Effort Multiplier</b>       | Specifies the relative time the Fitter spends in placement. The default value is 1.0, and legal values must be greater than 0. Specifying a floating-point number allows you to control the placement effort. A higher value increases CPU time but may improve placement quality. For example, a value of '4' increases fitting time by approximately 2 to 4 times but may increase quality.                                                                                                                                                               |
| <b>Power Optimization During Fitting</b> | Directs the Fitter to perform optimizations targeted at reducing the total power devices consume. The available settings for power-optimized fitting are: <ul style="list-style-type: none"> <li>• <b>Off</b>—performs no power optimizations.</li> <li>• <b>Normal compilation</b>—performs power optimizations that are unlikely to adversely affect compilation time or design performance.</li> <li>• <b>Extra effort</b>—performs additional power optimizations that might affect design performance or result in longer compilation time.</li> </ul> |

**Table 67. Advanced Fitter Settings (7 of 8)**

| Option                                                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Programmable Power Maximum High-Speed Fraction of Used LAB Tiles</b> | Sets the upper limit on the fraction of the high-speed LAB tiles. Legal values must be between 0.0 and 1.0. The default value is 1.0. A value of 1.0 means that there is no restriction on the number of high-speed tiles, and the Fitter uses the minimum number needed to meet the timing requirements of your design. Specifying a value lower than 1.0 might degrade timing quality, because some timing critical resources might be forced into low-power mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Programmable Power Technology Optimization</b>                       | Controls how the Fitter configures tiles to operate in high-speed mode or low-power mode. The following options are available: <ul style="list-style-type: none"> <li>• <b>Automatic</b>—specifies that the Fitter minimizes power without sacrificing timing performance.</li> <li>• <b>Minimize Power Only</b>—specifies that the Fitter sets the maximum number of tiles to operate in low-power mode.</li> <li>• <b>Force All Used Tiles to High Speed</b>—specifies that the Fitter sets all used tiles to operate in high-speed mode.</li> <li>• <b>Force All Tiles with Failing Timing Paths to High Speed</b>—sets all failing paths to high-speed mode. For designs that meet timing, the behavior of this setting is similar to the <b>Automatic</b> setting.</li> </ul> <p>For designs that fail timing, all paths with negative slack are put in high-speed mode. This mode likely does not increase the speed of the design, and it may increase static power consumption. This mode may assist in determining which logic paths need to be re-designed to close timing.</p> <p><i>Note:</i> Not available for Intel Stratix 10 devices.</p> |
| <b>Regenerate Full Fit Reports During ECO Compiles</b>                  | Controls whether the Fitter report is regenerated during ECO compilation. By default, this option is set to <b>Off</b> . Turning it <b>On</b> regenerates the report at the cost of compilation time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Router Timing Optimization Level</b>                                 | Controls how aggressively the router tries to meet timing requirements. Setting this option to <b>Maximum</b> can increase design speed slightly, at the cost of increased compile time. Setting this option to <b>Minimum</b> can reduce compile time, at the cost of slightly reduced design speed. The default value is <b>Normal</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Run Early Place during compilation</b>                               | Enables the Early Place Fitter stage during full compilation. Turning on this setting may increase Fitter processing time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

**Table 68. Advanced Fitter Settings (8 of 8)**

| Option                             | Description                                                                                                                                                                                                                                                                         |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synchronizer Identification</b> | Specifies how the Compiler identifies synchronization register chain registers for metastability analysis. A synchronization register chain is a sequence of registers with the same clock with no fan-out in between, which is driven by a pin or logic from another clock domain. |

*continued...*

| Option                                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                              | <p>The following options are available:</p> <ul style="list-style-type: none"> <li>• <b>Off</b>—the Timing Analyzer does not identify the specified registers, or the registers within the specified entity, as synchronization registers.</li> <li>• <b>Auto</b>—the Timing Analyzer identifies valid synchronization registers that are part of a chain with more than one register that contains no combinational logic. Use the <b>Auto</b> setting to generate a report of possible synchronization chains in your design.</li> <li>• <b>Forced if Asynchronous</b>—the Timing Analyzer identifies synchronization register chains if the software detects an asynchronous signal transfer, even if there is combinational logic or only one register in the chain.</li> <li>• <b>Forced</b>—the Timing Analyzer identifies the specified register, or all registers within the specified entity, as synchronizers. Only apply the <b>Forced</b> option to the entire design. Otherwise, all registers in the design identify as synchronizers.</li> </ul> <p>The Fitter optimizes the registers that it identifies as synchronizers for improved Mean Time Between Failure (MTBF), provided that you enable <b>Optimize Design for Metastability</b>. If a synchronization register chain is identified with the <b>Forced</b> or <b>Forced if Asynchronous</b> option, then the Timing Analyzer reports the metastability MTBF for the chain when it meets the design timing requirements.</p> |
| <b>Treat Bidirectional Pin as Output Pin</b> | Specifies that the Fitter treats the bidirectional pin as an output pin, meaning that the input path feeds back from the output path.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Weak Pull-Up Resistor</b>                 | Enables the weak pull-up resistor when the device is operating in user mode. This option pulls a high-impedance bus signal to VCC. Do not enable this option simultaneously with the <b>Enable Bus-Hold Circuitry</b> option. The Fitter ignores this option if you apply to anything other than a pin.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## 6.11 Document Revision History

This document has the following revision history.

**Table 69. Document Revision History**

| Date                | Version | Changes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2017.11.06          | 17.1.0  | <ul style="list-style-type: none"> <li>• Added support for Intel Stratix 10 Hyper-Aware design flow, Hyper-Retiming, Fast Forward compilation, and Fast Forward Viewer.</li> <li>• Added Advanced HyperFlex Settings topic.</li> <li>• Added Retiming Restrictions and Workarounds topic.</li> <li>• Added statement about Fast Forward compilation support for retiming across RAM and DSP blocks.</li> <li>• Added Concurrent Analysis topic.</li> <li>• Added Analyzing Fitter Snapshots topic.</li> <li>• Added Compilation Dashboard Early Place stage control image.</li> <li>• Added Running late_place After Early Place topic.</li> <li>• Updated for latest Intel naming conventions.</li> </ul>                                                                                                                     |
| 2017.05.08          | 17.0.0  | <ul style="list-style-type: none"> <li>• Added reference to initial compilation support for Cyclone 10 GX devices.</li> <li>• Described concurrent analysis following Early Place.</li> <li>• Updated Compilation Dashboard images for Timing Analyzer, Report, Setting, and Concurrent Analysis controls.</li> <li>• Updated description for Auto DSP Block Replacement in Advanced Synthesis Settings.</li> <li>• Updated Advanced Fitter Settings for Allow Register Retiming, and for removal of obsolete SSN Optimization option.</li> <li>• Added Prevent Register Retiming topic.</li> <li>• Added Preserve Registers During Synthesis topic.</li> <li>• Removed limitation for <b>Safe State Machine</b> logic option.</li> <li>• Added references to Partial Reconfiguration and Block-Based Design Flows.</li> </ul> |
| <i>continued...</i> |         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

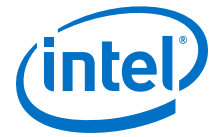


| Date       | Version | Changes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2016.10.31 | 16.1.0  | <ul style="list-style-type: none"> <li>Implemented Intel re-branding.</li> <li>Described Compiler snapshots and added Analyzing Snapshot Timing topic.</li> <li>Updated project directory structure diagram.</li> <li>Described new Fitter stage menu commands and reports.</li> <li>Added description of Early Place Flow, Implement Flow, and Finalize Flow.</li> <li>Added description of Incremental Optimization in the Fitter.</li> <li>Reorganized order of topics in chapter.</li> <li>Removed deprecated <b>Per-Stage Compilation (Beta)</b> Compilation Flow.</li> </ul> |
| 2016.05.03 | 16.0.0  | <ul style="list-style-type: none"> <li>Added description of Fitter Plan, Place and Route stages, reporting, and optimization.</li> <li>Added <b>Per-Stage Compilation (Beta)</b> Compilation Flow</li> <li>Added Compilation Dashboard information.</li> <li>Removed support for <b>Safe State Machine</b> logic option. Encode safe states in RTL.</li> <li>Added Generating Dynamic Synthesis Reports topic.</li> <li>Updated Quartus project directory structure.</li> </ul>                                                                                                    |
| 2015.11.02 | 15.1.0  | <ul style="list-style-type: none"> <li>First version of document.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

### Related Links

#### [Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 7 Block-Based Design Flows

---

The Intel Quartus Prime Pro Edition software supports block-based design flows, also known as modular or hierarchical design flows. These flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, as well as reuse of design blocks in other projects.

You can reuse design blocks with the same periphery interface, share a synthesized design block with another designer, or replicate placed and routed IP in another project. Design, implement, and verify core or periphery blocks once, and then reuse those blocks multiple times across different projects that use the same device.

In block-based design flows, you assign hierarchical instances of logic blocks in the design as *design partitions*. The design partition is a logical, named, hierarchical boundary assignment.

### Design Block Reuse

In design block reuse flows, you export a core or root partition for reuse in another project that targets the same Intel FPGA device. You can share specific compilation snapshots of the partitions with other projects and designers, such as synthesized, placed, or final snapshot.

Root partition reuse enables preservation of compilation results for a top-level (or *root*) partition that describes the device periphery, along with associated core logic. Reuse of the periphery allows a board developer to create and optimize a platform design with device periphery logic once, and then share that root partition with other board users who create custom core logic. The periphery resources include all the hardened IP in the device periphery, such as general purpose I/O, PLLs, high-speed transceivers, PCIe, and external memory interfaces.

Team members can work on different partitions separately, and then bring them together later, facilitating a team-based design environment. A *team lead* integrates the partitions in the system and provides guidance to ensure that each partition uses the appropriate device resource and achieves design requirements during the full design integration. A partition *Developer* initially develops and exports a block as a partition in one Intel Quartus Prime Developer project. Subsequently, a partition *Consumer* reuses the partition in a different Consumer project.<sup>(6)</sup> To avoid resource conflicts, floorplanning is essential when reusing placed or routed partitions.

---

<sup>(6)</sup> For brevity, this document uses Developer to indicate the person or project that originates a reusable block, and uses Consumer to indicate the person or project that consumes a reusable block.



## Incremental Block-Based Compilation

The incremental block-based compilation flow enables you to preserve or empty a partition that contains FPGA core logic. FPGA core resources include LUTs, registers, memory blocks, and DSP blocks. Core partitions can include only core resources, and cannot include any periphery resources.

Incremental block-based compilation can improve the predictability of results during design iterations. You can preserve partitions at the synthesis, placement, or final snapshot. The preserved partitions remain at the same preservation level in subsequent compilations. The Compiler modifies only the non-preserved partitions in the design.

You can target optimization techniques to specific design partitions, while leaving other partitions unchanged. When you preserve the compilation results for a partition, the preserved partition database acts as the source for subsequent compilations.

You can use empty partitions to represent parts of your design are incomplete or missing, while you compile the rest of your design. Setting a partition to **Empty** can also reduce the compilation time for the other parts of the design because the Compiler does not process design logic associated with an empty partition.

### Related Links

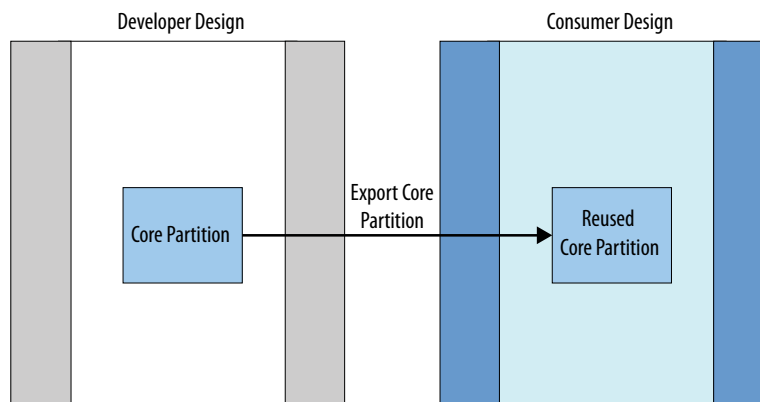
- [Design Compilation](#)
- [Creating a Partial Reconfiguration Design](#)

## 7.1 Block-Based Design Examples

### Core Block Reuse Example

In a typical core partition reuse example, a Developer preserves and reuses a core partition that already meets design requirements, or reuses a core partition that another Developer designs. The Developer optimizes and preserves the block, and then the Consumer can simply reuse the block without requiring re-optimization in the Consumer project.

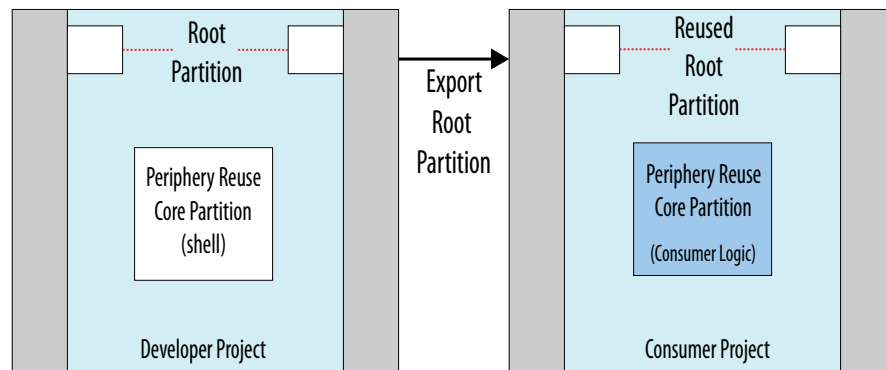
Figure 98. Core Reuse Example



### Periphery Reuse Example

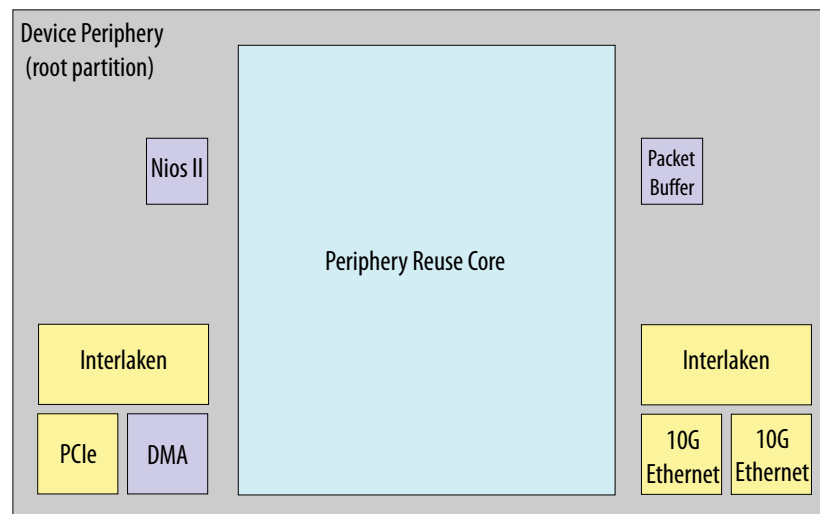
In a typical periphery reuse example, a Developer defines a root partition that includes periphery and core resources that are appropriate for reuse in other projects. An example of this scenario is a development kit that multiple Developers and projects can use.

**Figure 99. Root Partition Reuse Example**



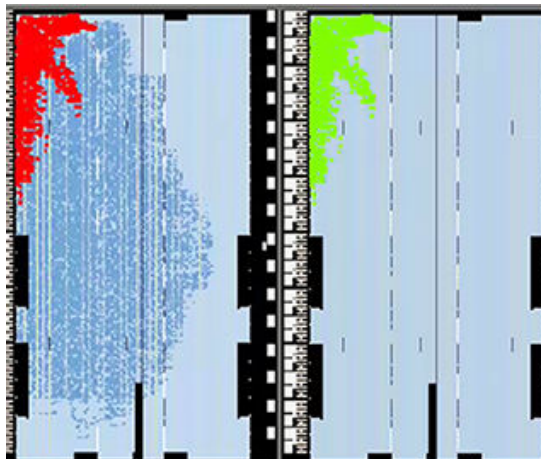
In root partition reuse, each project must target the same Intel FPGA part number, must have the same interfaces, and use the same version of the Intel Quartus Prime Pro Edition software. The following example shows reuse of an optimized root partition that contains various periphery interfaces. Only the periphery reuse core partition that contains custom logic changes between Consumer projects.

**Figure 100. Periphery Reuse Example**



You can preserve a block with unique characteristics that you want to retain, and then replicate that functionality or physical implementation in other projects. In the following figure, a Developer reuses the red-colored partition in the floorplan in another project shown in green in the floorplan on the right.

**Figure 101. IP Replication and Physical Implementation**



## 7.2 Design Partitioning

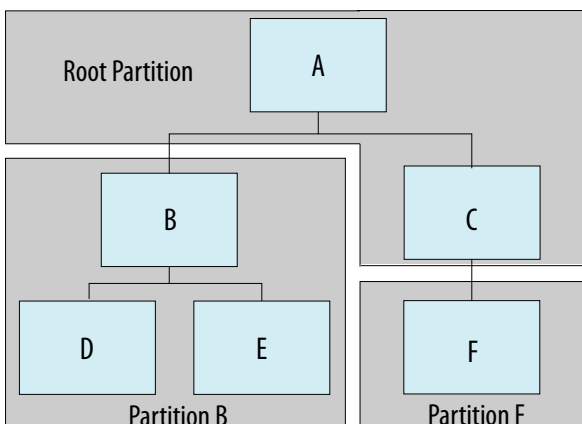
To use block-based design flows, you must first create design partitions from your design's hierarchical instances. The Intel Quartus Prime Compiler then treats the design partitions separately to allow the block-based functionality.

When you define partitions, every hierarchy within that partition becomes part of the parent partition. When you create child partitions for hierarchies within an existing partition, the logic within the new child partition is no longer part of the parent partition.

By default, every Intel Quartus Prime project includes a single, root partition. The root partition contains all the periphery resources, and may also include core resources. When you export the root partition for reuse, the exported partition includes all logic that you do not include in periphery reuse core partitions. Therefore, to export and reuse periphery elements, you export the root partition.

**Figure 102. Design Partitions in Design Hierarchy**

In the following example, instances B and F are designated design partitions. Partition B includes sub-instances D and E. The root partition contains the top-level instance A and instance C, because C is unassigned to any partition.



Design partitions facilitate incremental block-based compilation and design block reuse by logically separating instances. This logical separation allows the Compiler to synthesize and optimize each partition separately from the other parts of the design. The logical separation can also prevent Compiler optimizations across partition boundaries.

### 7.2.1 Planning Design Partitions

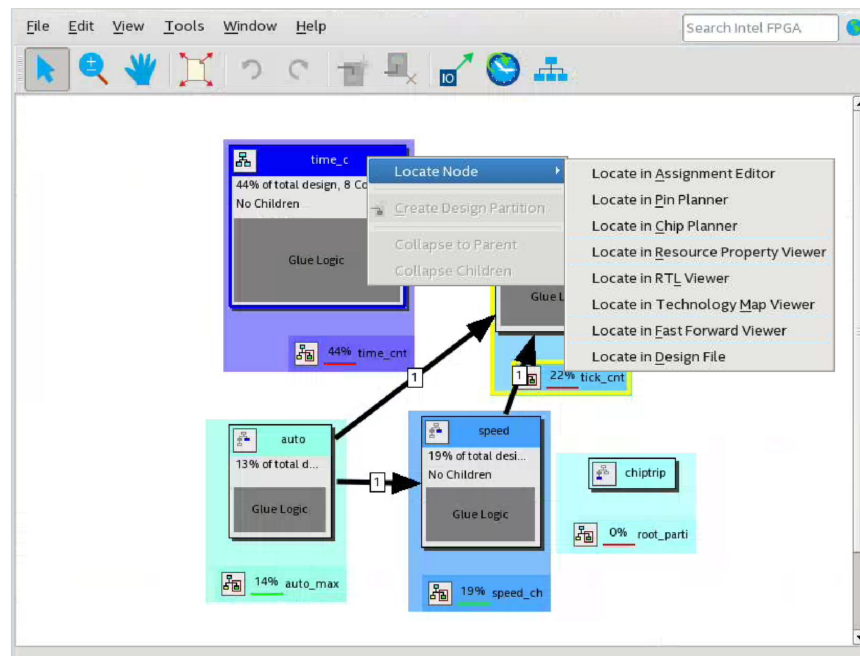
Block-based design flows require more up-front planning than full flat compilation. For example, you must structure the source code and design hierarchy to ensure proper logic grouping for optimization. Implementing the correct logic grouping is easiest early in the design cycle.

Creating or removing a design partition changes the synthesis and subsequent physical implementation and quality of results. When planning the design hierarchy, be aware of the size and scope of each partition, and the possibility of different parts of the design changing during development. Separate logic that changes frequently from the fixed parts of the design.

Group design blocks in your design hierarchy so that highly-connected blocks have a shared level of design hierarchy for assignment to one partition. Structuring your design hierarchy appropriately reduces the required number of partition boundaries, and allows maximum optimization within the partition.

The Design Partition Planner (**Tools > Design Partition Planner**) helps you to visualize and refine a design's partitioning scheme by showing timing information, relative connectivity densities, and the physical placement of partitions. You can locate partitions in other viewers, or modify or delete partitions in the Design Partition Planner.

Figure 103. Design Partition Planner







Consider creating each design entity that represents a partition instance in a separate source file. This approach helps you correlate which partitions require recompilation, instead of reusing preserved results, when you make source code changes. As you make design changes, you can designate partitions as empty or preserved to instruct the Compiler which partitions to recompile from source code, as [Creating and Modifying Design Partitions](#) on page 250 describes.

If your design has timing-critical partitions that are changing through the design flow, or partitions exported from another project, use design floorplan assignments to constrain the placement of the affected partitions. A properly partitioned and floorplanned design enables partitions to meet top-level design requirements when you integrate the partitions with the rest of your design. Poorly planned partitions or floorplan assignments negatively impact design area utilization and performance, thereby increasing the difficulty of timing closure.

The following design partition guidelines help ensure the most effective and efficient results. Block-based design flows add steps and requirements to the design process, but can provide significant benefits in design productivity.

#### Related Links

- [Interface Planning](#)
- [Design Floorplan Analysis in the Chip Planner](#)

### 7.2.1.1 Planning Core and Root Partitions

By default, every Intel Quartus Prime project has a single, root partition. The root partition contains all the periphery resources, and may also include core resources. Each project can include multiple core partitions.

#### Planning Partitions for Periphery IP

- Plan the design periphery to segregate and implement periphery resources in the root partition. Ensure that IP blocks that utilize both core and periphery resources (such as transceiver and external memory interface Intel FPGA IP) are part of the root partition.
- When creating design partitions for an existing design, remove all periphery resources from any entity you want to designate as a core partition. Also, tunnel any periphery resource ports to the top level of the design. Implement the periphery resource in the root partition.
- You cannot designate instances that use periphery resources as separate partitions. In addition, you cannot split an Intel FPGA IP core into more than one partition.
- The Intel Quartus Prime software generates an error if you include periphery interface Intel FPGA IP cores in any partition other than the top-level root partition.
- You must include Intel FPGA IP cores for the Hybrid Memory Cube (HBM) or Hard Processor System (HPS) in the root partition.

### Planning Partitions for Clocks and PLLs

- Plan clocking structures to keep all PLLs and corresponding clocking logic in the root partition. This technique allows the Compiler to control PLLs in the root partition, if necessary.
- Consider creating a design block for all clocking logic that you instantiate in the top-level of the design. This technique ensures that the Compiler groups clocking logic together, and that the Compiler treats clocking logic as part of the root partition. Clock routing resources belong to the root partition, but the Compiler does not preserve routing resources with a partition.
- Include any signal that you want to drive globally in the root partition, rather than the core partition. Signals (such as clocks or resets) that you generate inside core partitions cannot drive to global networks without a clock buffer in the root partition.
- To support existing Intel Arria 10 designs, the Compiler allows I/O PLLs in core partitions. However, creating a partition boundary prevents such PLLs from merging with other PLLs. The design may use more PLLs without this merging, and may have suboptimal clocking architecture.

#### 7.2.1.2 Design Partition Guidelines

Designating a hierarchical design instance as a design partition creates a logical hierarchical boundary around that instance. This partition boundary limits the Compiler's ability to merge the partition's logic with other parts of the design. A partition boundary can also prevent optimization that reduces cell and interconnect delay, which can reduce design performance. Follow these guidelines when creating your design hierarchy and assigning partitions:

- Register partition boundary ports. This practice can reduce unnecessary long delays by confining register-to-register timing paths to a single partition for optimization. This technique also minimizes the effect of the physical placement for boundary logic that the Compiler might place without knowledge about the other partition.
- Minimize the timing-critical paths passing in or out of design partitions. For timing critical-paths that cross partition boundaries, rework the partition boundaries to avoid these paths. Isolate timing-critical logic inside a single partition, so the Compiler can effectively optimize each partition independently.
- Avoid creating a large number of small partitions throughout the design. Excessive partitioning can impact performance by preventing design optimizations.
- Avoid grouping unrelated logic into a large partition. If you are working to optimize an independent block of your design, assigning that block as a small partition provides you more flexibility during optimization.
- When using incremental block-based design within a single project, the child partition must have an equal or higher preservation level than the parent. If the parent partition has a higher preservation level than the child, the Compiler ignores the preservation level.

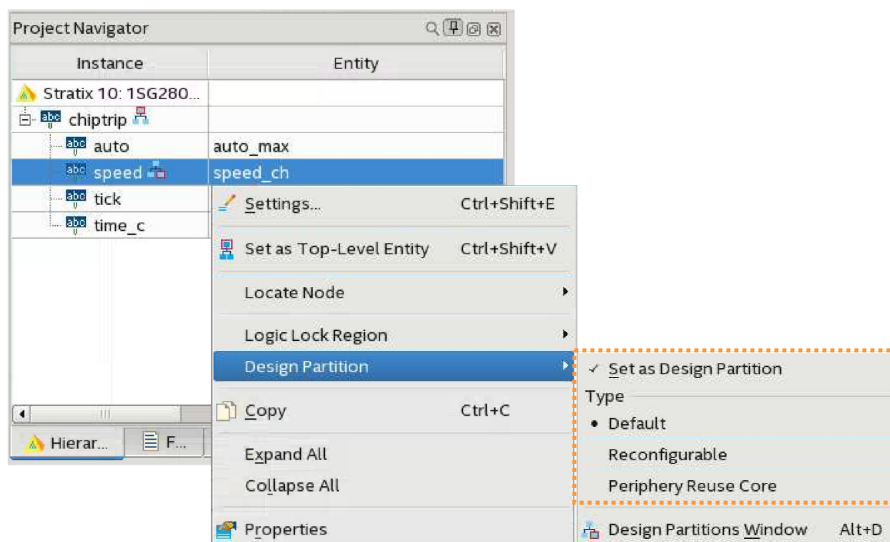
#### 7.2.2 Creating and Modifying Design Partitions

A design partition is a logical, named, hierarchical boundary assignment. You can create a partition that preserves the results of the synthesized, placed, or final compilation snapshot. Before creating a partition you must elaborate the design hierarchy. You can view and modify all design partitions for a project in the Design



Partitions Window. The Compiler assigns a default partition name, which you can edit. All design partition names must be unique, and can consist of only alphanumeric and underscore ( \_ ) characters. Follow these steps to create and modify design partitions:

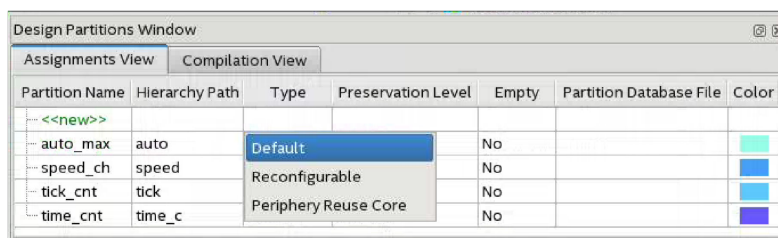
1. Click **Processing > Start > Start Analysis & Elaboration**.
2. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition > Set as Design Partition**. A design partition icon appears next to each instance you assign.



This setting corresponds to the following assignment in the .qsf:

```
set_instance_assignment -name PARTITION <name> \
    -to <partition hierarchical path>
```

3. To view and edit all design partitions, click **Assignments > Design Partitions Window**. You can also define new partitions in this window.
4. To specify a partition **Type**, double-click the **Type** for a partition, and then select **Default**, **Reconfigurable**, or **Periphery Reuse Core**.

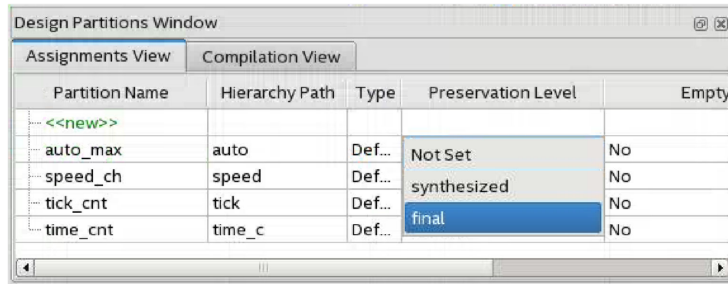


This setting corresponds to the following assignment in the .qsf:

```
set_instance_assignment -name PARTITION <name> -to \
    <partition hierarchical path>
```

5. If you specify the **Default** partition **Type**, specify one or more of the following options according to the type of information you want to preserve in the partition. If you specify a partition **Type** other than **Default**, retain the default settings for these options:

- To preserve the compilation results for a partition, double-click the **Preservation Level** column for a partition, and then select **synthesized** or **final**. Setting the **Preservation Level** to **Not Set** indicates no preservation.



This setting corresponds to the following assignment in the .qsf:

```
set_instance_assignment -name PRESERVE <FINAL|SYNTHESIZED> \
    -to <partition hierarchical path>
```

- To specify a Partition Database File (.qdb) for design reuse, double-click the **Partition Database File** column for a partition, and then browse to the appropriate .qdb file for your partition. To remove a .qdb from subsequent compilation, remove the **Partition Database File** setting.
6. View the partition names, hierarchy paths, and instances after compilation in the **Compilation View** tab. The synthesis and final partitions reports provide additional information about preservation levels and .qdb file assignments.

The screenshot shows the 'Design Partitions Window' with the 'Compilation View' tab selected. The table displays resource usage for various partitions, including the root partition and sub-partitions like auto\_max, speed\_ch, tick\_cnt, and time\_cnt. The columns are Partition Name, Hierarchy Path, Combinational ALUTs, and Dedicated Logic Registers.

| Partition Name | Hierarchy Path | Combinational ALUTs | Dedicated Logic Registers |
|----------------|----------------|---------------------|---------------------------|
| root_partition |                | 21                  | 19 / 3732480 (< 1 %)      |
| auto_max       | auto           | 6                   | 3 / 3732480 (< 1 %)       |
| speed_ch       | speed          | 3                   | 4 / 3732480 (< 1 %)       |
| tick_cnt       | tick           | 4                   | 4 / 3732480 (< 1 %)       |
| time_cnt       | time_c         | 8                   | 8 / 3732480 (< 1 %)       |

*Note:* Refer to [Define Empty Partitions to Reduce Compilation Time](#) on page 256 for information about empty partitions.

### 7.2.3 Defining an Empty Partition

You can define an empty partition for core partitions that are incomplete, or that you want to ignore during compilation. When a core partition is set to **Empty**, the Compiler ties the partition output ports off to ground, and removes the input ports.

The Compiler removes any existing synthesis, placement, and routing information for an empty partition. If you remove the **Empty** setting from a partition, the Compiler re-implements the partition from the source.



To define an empty partition, follow these steps in the Design Partitions Window:

1. Set the partition **Type** to **Default**. Any other setting is incompatible with empty partitions.
2. Set the **Preservation Level** to **Not Set**. If you set the **Preservation Level** to **Synthesized** or **Final**, the Compiler ignores the **Empty** option.
3. Set the **Empty** option to **Yes**.

This setting corresponds to the following assignment in the .qsf:

```
set_instance_assignment -name EMPTY ON -to \  
<hierarchal path of partition> -entity <name>
```

4. Remove any .qdb specification for the **Partition Database File** option.

## 7.2.4 Top-Down, Bottom-Up, and Team-Based Design Methods

In top-down hierarchical design methodologies, you plan the design at the top level, and then split the design into lower-level design blocks. Different developers or intellectual property (IP) providers can create and verify HDL code for lower-level design blocks separately, but one team lead manages the implementation project for the entire design.

In the traditional concept of bottom-up design, you create lower-level design blocks independently of one another, and then integrate the blocks at the top-level. To implement a bottom-up design flow, individual developers or IP providers can complete the placement and routing optimization of their design in separate projects, and then reuse each lower-level block in one top-level project. These methodologies can be useful for team-based design flows with developers in other locations, or when third-parties create the design partitions.

However, when developing design blocks independently in a bottom-up design flow, individual developers may initially lack information about the overall design, or how their block connects with other blocks. This lack of information can lead to problems during system integration, such as difficulties with timing closure, or resource conflicts. To reduce such difficulties, plan the design at the top level, whether optimizing within a single project, or optimizing blocks independently in separate projects for subsequent top-level integration.

You can meet some of the goals of a bottom-up flow with a design that is planned top-down if you need to compile and optimize design blocks, when other parts of the design are missing or incomplete. Designate missing or incomplete design blocks as empty partitions, as [Defining an Empty Partition](#) on page 252 describes.

### 7.2.4.1 Team-Based Design

You can use elements of both top-down and bottom-up design methodologies to implement a successful team-based design flow.

A top-level design can include one or more partitions that different designers or IP providers create and optimize, as well as partitions for development as part of a standard incremental block-based design compilation. In a team-based environment, portions of your design may initially be undefined with the expectation of developing those portions later. The team lead or system architect creates empty partitions in the top-level design for any incomplete partitions. Developers or IP providers can then create and verify HDL code separately, and the team lead later integrates the code into the single Consumer project.



For easiest integration of partitions, add the completed partitions to the design incrementally, and confine the optimization to the top-level design. Use a single Intel Quartus Prime project for the partition integration, whenever possible. Using multiple projects for integration can add significant debugging time.

If you cannot use a single project, you can create the partition in a copy of the top-level Intel Quartus Prime project, and then export the completed partition. The team lead then integrates each design block as a design partition into the top-level design. To simplify full design optimization by allowing full-chip placement and routing of the partition at the top-level, export and reuse only the synthesis snapshot, unless the top-level design requires optimized post-fit results.

Teams that use a bottom-up design method can optimize placement and routing of design partitions independently. However, the following drawbacks can occur when optimizing the design partitions in separate projects:

- Achieving timing closure for the full design may be more difficult if you compile partitions independently without information about other partitions in the design. Avoiding this problem requires careful manual timing budgeting and observance of design rules, such as always registering the ports at the module boundaries.
- The design requires resource budgeting and allocation to avoid resource conflicts and overuse. Floorplanning with Logic Lock regions can help you avoid resource conflicts while developing each part independently in a separate Intel Quartus Prime project.
- Maintaining consistency of assignments and timing constraints is more difficult if you use separate Intel Quartus Prime projects. The team lead must ensure that the assignments and constraints of the top-level design, and those Developers define in the separate projects and reuse at the top-level, are consistent.

Partitions that you develop independently all must share a common set of resources. To minimize issues that can arise when sharing a common set of resources between different partitions create the partitions within in a single project, or in copies of the top-level project, with full design-level constraints, to ensure that resources do not overlap. Correct use of partitions and Logic Lock regions can help to minimize issues that can arise when integrating into the top-level design.

The use of a common project ensures that each Developer has a consistent view of the top-level design framework. When another Developer provides and optimizes timing-critical portions of the design, each Developer must have the complete top-level design framework to maintain timing closure and achieve the best results during integration.

If a Developer has no information about the top-level design, the team lead must at least communicate a specific Intel FPGA device part number, along with any physical timing constraints. The Developer can then create and export the partition from a separate project. When a Developer lacks information, it is prudent to overconstrain or create additional timing margin on the individual partitions. The technique reduces the chance of timing problems when integrating the partitions with other blocks.

### Related Links

[Creating a Top-Level Project for a Team-Based Design](#) on page 269

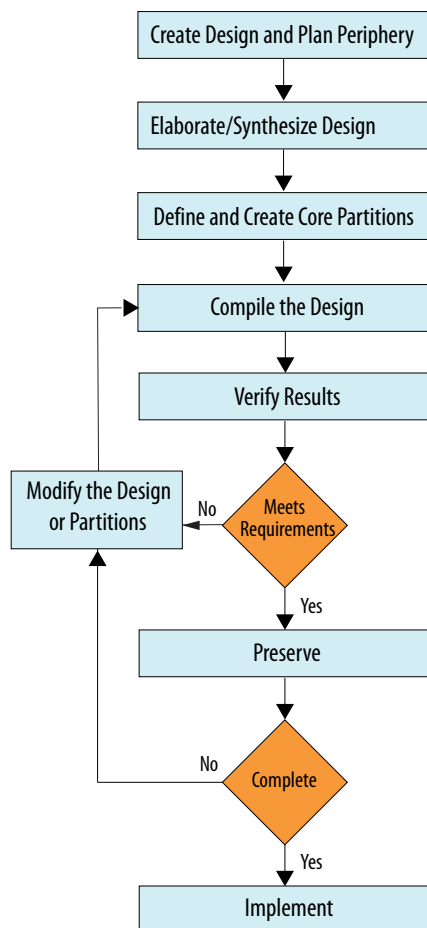


## 7.3 Incremental Block-Based Compilation

Incremental block-based compilation flows allow you to iteratively preserve partitions as they meet your design requirements, which provides more predictable results. After initial project setup, define the design periphery in the top-level hierarchy of the design. Next, create a design hierarchy that allows you to create the core partitions that your design requires.

After you define the initial project structure and add source design files, run **Analysis & Elaboration** to display the project **Hierarchy** in the Project Navigator and begin defining partitions. Following design compilation, analyze partitions for specific results, such as timing closure, or resource utilization. When the partition meets the design requirements, you can preserve the partitions at the synthesized, placed, or final stage, depending on the requirements of the design.

**Figure 104. Incremental Block-Based Design Flow**



### Related Links

[Design Partition Guidelines](#) on page 250

### 7.3.1 Define Empty Partitions to Reduce Compilation Time

You can define empty partitions to reduce compilation time in incremental block-based compilation flows. Setting a partition to **Empty** reduces design compilation time because the top-level design netlist does not include the partition design logic. Therefore, the Compiler does not run full synthesis and fitting algorithms on the empty partition logic.

To save the compilation results for a partition before setting to **Empty**, export the `.qdb` for the partition, and then reuse the `.qdb` file when you no longer require the empty partition. Emptying a preserved partition removes all preserved information.

**Note:**

The following guidelines apply to empty partitions:

1. Do not set a partition that has a `.qdb` assignment to **Empty**. To empty a partition that has a `.qdb` assignment, first remove the `.qdb` file specification in the Design Partitions Window.
2. You may want to create a **Reserved** Logic Lock region constraint for empty partitions to avoid resource conflicts with other partitions. This constraint can help avoid fragmentation of the partition upon reintegration with the top-level design. This constraint ensures that the Compiler does not place other logic in the area that you reserve for the empty partition.

#### Related Links

[Design Block Reuse](#) on page 256

## 7.4 Design Block Reuse

Design block reuse allows you to preserve a design partition as an exported `.qdb` file, and reuse this partition in another project. Reuse of core or root partitions involves partitioning and constraining the block prior to compilation, exporting, and reusing the block. Effective design block reuse requires careful planning to ensure that the source code and design hierarchy support the physical partitioning of device resources that these flows require.

- **Core partition reuse**—allows reuse of synthesized, placed, or final snapshots of design blocks in another project.
- **Root partition reuse**—allows reuse of a synthesized, placed, or final snapshots of the root partition. The root partition includes periphery resources (including I/O, HSSIO, PCIe, PLLs), as well as any associated core resources, while leaving a core partition open for subsequent development.

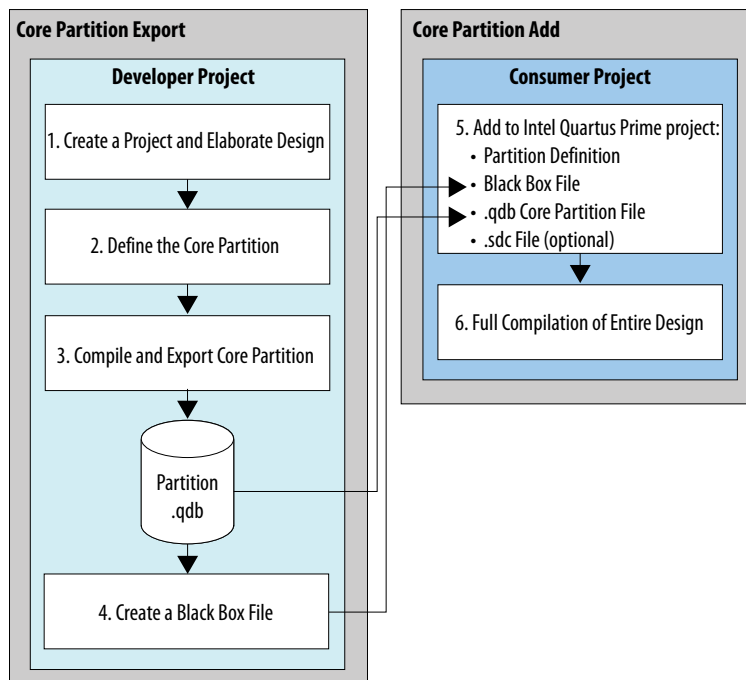
At a high level, the core and root partition reuse flows are similar. Both flows preserve and reuse a design partition as a `.qdb` file. The design block Developer defines, compiles, and preserves the block in the Developer project, and the Consumer reuses the block in one or more Consumer projects.



## 7.4.1 Reusing Core Partitions

Reusing core partitions involves exporting the core partition from the Developer project as a .qdb, and then reusing that core partition in a different Consumer project. You assign the .qdb to an instance in the Consumer project. In the Consumer project, the Compiler only runs Analysis and other compilation stages that occur after the stage you preserve in the .qdb from the Developer project.

Figure 105. Core Partition Reuse Flow



The following steps describe the core partition reuse flow in detail.

### 7.4.1.1 Step 1: Developer: Define a Core Partition

Define design partitions to create logical boundaries in the design hierarchy. Confine each core instance for export within a design partition. You can define partition instances from the Project Navigator or in the Design Partitions Window.

To define a core design partition:

1. Review the project to determine design elements suitable for reuse, and the appropriate snapshot for export.
2. Follow the steps in [Creating and Modifying Design Partitions](#) on page 250 to define a core partition. Select **Default** for the partition **Type**.

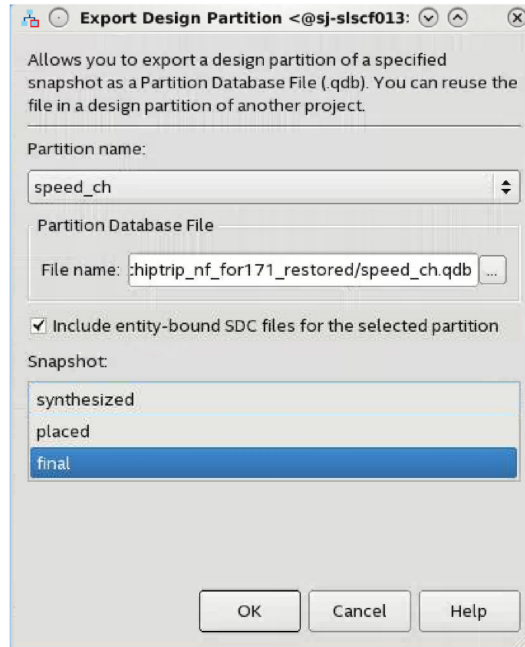
### 7.4.1.2 Step 2: Developer: Compile and Export a Core Partition

This step describes generating a final snapshot for export. Following compilation, you can export a partition as a .qdb at the synthesized, placed, or final stage. You can then reuse the core partition in the same project or in another project.

To compile and export a core partition

1. To run all compilation stages through Fitter (Finalize) and generate the final snapshot, click **Processing > Start > Start Fitter**.
2. To export the core partition, click **Project > Export Design Partition**. Select the **Design Partition** name and the compilation **Snapshot** for export.
3. Confirm the **File name** for the **Partition Database File**, and then click **OK**.

**Figure 106. Export Design Partition**



The following command corresponds to partition export in the GUI:

```
quartus_cdb <project name> -c <revision name> \
--export_partition "<name>" --snapshot <synthesized|placed|final> \
--file <name>.qdb --preserve_sdc
```

### 7.4.1.3 Step 3: Developer: Create a Black Box File

Integrating a core partition .qdb file also requires that you add a supporting black box file to the consumer project. The black box file defines the ports and port interface types for synthesis in the Consumer project. Follow these steps to create a block box port definitions file for the partition.

1. Create an HDL file (.v, .vhdl, .sv) that contains only the port definitions for the exported core partition. Include parameters passed to the module. For example:

```
module bus_shift #(
    parameter DEPTH=256,
    parameter WIDTH=8
) (
    input clk,
    input enable,
    input reset,
    input [WIDTH-1:0] sr_in,
```



```
output [WIDTH-1:0] sr_out
);
endmodule
```

2. Provide the black box file and core partition .qdb file to the Consumer.

#### 7.4.1.4 Step 4: Consumer: Add the Core Partition and Compile

To add the core partition to the Consumer project, add the black box as a source file in the project, and assign the core partition .qdb to an instance in the Design Partitions Window. Because the exported .qdb includes compiled netlist information, the Consumer project must target the same FPGA device part number and use the same Intel Quartus Prime version as the Developer project. The Consumer project must supply a clock and any other constraints required for the interface to the core partition.

1. Create or open an Intel Quartus Prime project to reuse the core partition.
2. To add one or more black box files to the consumer project, click **Project > Add/Remove Files in Project** and select these files.
3. Follow the steps in [Creating and Modifying Design Partitions](#) on page 250 to define core partition for the black box file.
4. To run all compilation stages through Fitter (Finalize) and generate the final snapshot, click **Processing > Start > Start Fitter**.
5. The Fitter Partition Summary report lists partition information, such as the partition name, hierarchy path, snapshot preservation level, and any associated .qdb file.

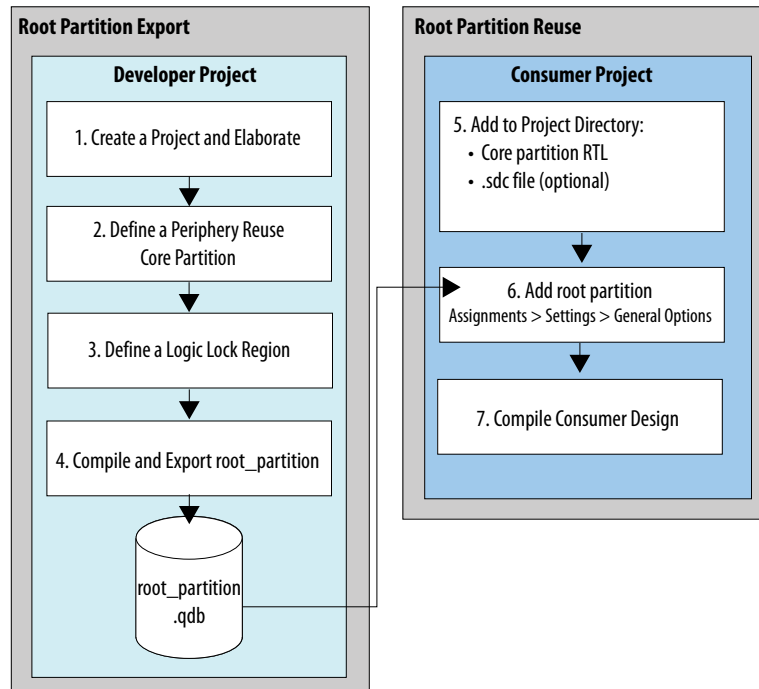
Figure 107. Fitter Partition Summary Report

|   | Partition Name | Hierarchy Path | Preservation | Empty | Partition Database File |
|---|----------------|----------------|--------------|-------|-------------------------|
| 1 | root_partition |                |              |       |                         |
| 2 | speed_ch       | speed          |              |       |                         |

#### 7.4.2 Reusing Root Partitions

The root partition contains all the periphery resources, and may also include some core resources. To export and reuse periphery elements, you export the root partition. Reuse of root partitions allows you to design an FPGA-to-board interface and associated logic once, and then replicate that interface in other projects.

Figure 108. Root Partition Reuse Flow



#### 7.4.2.1 Step 1: Developer: Create a Periphery Reuse Core Partition

To export and reuse the root partition, first create a periphery reuse core partition for later core logic development in the Consumer project.

To create a periphery reuse core partition:

1. Follow the steps in [Creating and Modifying Design Partitions](#) on page 250 to define a periphery reuse core partition.
2. When defining the partition, select **Periphery Reuse Core** for the partition **Type**.

#### 7.4.2.2 Step 2: Developer: Define a Logic Lock Region

You must define a core-only, reserved, fixed routing region to reserve core resources in the Consumer project for the periphery reuse core partition. The Consumer uses this area for non-periphery development, and the area can contain only core logic. Ensure that the exclusive placement region size can contain all core logic. For projects with multiple core partitions, constrain each partition in a non-overlapping routing region.



Follow these steps to define a core-only, reserved, fixed routing region to reserve core resources in the Developer project for non-periphery development:

1. Right-click the design instance in the **Project Navigator** and click **Logic Lock Region > Create New Logic Lock Region**. The region appears in the Logic Lock Regions Window. You can also verify the region in the Chip Planner (**Locate Node > Locate in Chip Planner**).
2. Specify the placement region co-ordinates in the **Origin** column.
3. Enable the **Reserved** and **Core-Only** options.
4. Click the **Routing Region** cell. The **Logic Lock Routing Region Settings** dialog box appears.

Specify the Routing Region Type and Expansion Length

Specify Core-Only as On

| Region Name        | Member | Width | Height | Origin    | Reserved | Core-Only | Size/State   | Routing Region         |
|--------------------|--------|-------|--------|-----------|----------|-----------|--------------|------------------------|
| Logic Lock Regions |        |       |        |           |          |           |              |                        |
| u_blinking_led     | accel  | 20    | 20     | X169_Y140 | On       | On        | Fixed/Locked | Fixed with expansion 1 |
| <<new>>            |        |       |        |           |          |           |              |                        |

Specify Height and Width

Specify Origin Coordinates

Specify Reserved as On

5. Specify **Fixed with expansion** with **Expansion Length** of **0** for the **Routing Type**.
6. Click **OK**.
7. Click **File > Save Project**.

### 7.4.2.3 Step 3: Developer: Compile and Export the Root Partition

After compilation, you can export the root partition at the synthesized, placed, or final stage. You can optionally supply any Synopsys Design Constraints (.sdc) file for the partition you export. If you supply an .sdc with the partition, the Consumer project uses the file for evaluation for all snapshots. In addition, the Consumer project uses the .sdc to drive placement and routing for synthesis snapshots, and to drive routing for placed snapshots.

1. To run all compilation stages through Fitter (Finalize), click **Processing > Start > Start Fitter**.
2. To export the root partition to a .qdb file, click **Project > Export Design Partition**. Select the **root\_partition** and the **synthesized, placed, or final** snapshot. For Intel Stratix 10 designs, the **Include entity-bound SDC files for the selected partition** option is on by default.

The following command corresponds to root partition export in the GUI:

```
quartus_cdb <project name> -c <revision name> \
--export_partition "root_partition" --snapshot final \
--file root_partition.qdb --preserve_sdc
```

3. The Developer provides the exported .qdb file, any optional .sdc files, and black box file for the periphery reuse core to the Consumer.

#### 7.4.2.4 Step 4: Consumer: Add the Root Partition and Compile

To add the root partition to a Consumer project, you assign the exported `root_partition.qdb` to the project in the New Project Wizard, or by clicking **Assignments > Settings > General**. In addition, you must add the core partition RTL, `.ip`, or `.qsys` files as project source files. The `root_partition.qdb` includes all Logic Lock constraints from the Developer project. There is no need to recreate these constraints in the Consumer project. After assigning the `.qdb`, the Consumer project includes all additional information from the Developer project compilation snapshot, including synthesis, placement, or final compilation results.

Follow these steps to reuse the root partition to a Consumer project:

1. The Consumer obtains from the Developer the exported `.qdb` file, any optional `.sdc` files, and a port list for the periphery reuse core.
2. Click **File > New Project Wizard** to create an Intel Quartus Prime project to reuse the exported root partition.
3. On the **Directory, Name, and Top-Level Entity** page, enable **This project uses a Partition Database (.qdb) file for the root partition**, and specify the `.qdb` file for root partition.
4. On the **Add Files** wizard page, add all design-level `.sdc` files from the Developer project as source files in the Consumer project. For Intel Arria 10 designs only, also optionally add the `.sdc` files for Intel FPGA IP cores to the Consumer project.

*Note:* For Intel Stratix 10 designs, the **Include entity-bound SDC files for the selected partition** option is enabled by default, and there is no requirement to separately add the IP core `.sdc` files to the Consumer project. All Intel FPGA IP cores use entity-bound `.sdc` files for Intel Stratix 10 designs. This connection enables automatic bundling of the `.sdc` that you use in the partition `.qdb` file.

5. Specify the remaining settings in the wizard and click **Finish**.
6. To run all compilation stages, click **Processing > Start Compilation**. The Compiler implements the reused root partition and constraints.

## 7.5 Debugging Block-Based Designs

You can use the Signal Tap logic analyzer to debug block-based designs. The following section describes Signal Tap debugging of designs that include reusable blocks.

When reusing core blocks, you can expose potential Signal Tap nodes to partition boundary ports for tapping the block in a Consumer project. In the core block reuse flow, the Developer creates the partition boundary ports for each point that Signal Tap uses. All boundary ports that the Developer creates are then available in the Consumer project for debugging.

In addition, when reusing periphery blocks, the Developer and Consumer instantiate a debug bridge to extend Signal Tap debugging into the core partition. The Consumer of a root partition can then use Signal Tap to debug in the periphery reuse core partition by connecting to this debug bridge. To use this bridge, you must instantiate an SLD JTAG Bridge Agent and Host pair for each periphery reuse core boundary in your design, as [SLD JTAG Bridge Intel FPGA IP](#) on page 267 describes. You instantiate the



SLD JTAG Bridge Agent in the parent partition, and the SLD JTAG Bridge Host in the child partition. You can then connect to a Signal Tap instance through the bridge via the bridge agent and host pair.

For more information about the SLD JTAG bridge instantiation, refer to *Instantiating a SLD JTAG Bridge Agent* and *Instantiating a SLD JTAG Bridge Host* in the Intel Quartus Prime Pro Edition handbook.

#### Related Links

- [Debugging Designs with the Signal Tap Logic Analyzer](#)
- [Introduction to Intel FPGA IP Cores](#)
- [Instantiating a SLD JTAG Bridge Agent](#)
- [Instantiating a SLD JTAG Bridge Host](#)

### 7.5.1 Signal Tap with Core Partition Reuse

The Intel Quartus Prime software supports different methods for using Signal Tap to debug of core partitions, depending on usage of the partition's .qdb file. The following sections describe each core partition debugging method.

#### 7.5.1.1 Using HDL Signal Tap Instances

You can instantiate HDL Signal Tap instances in reusable core partitions without any additional steps. Simply instantiate the Signal Tap component in the HDL of the Developer and Consumer projects and compile.

When using HDL Signal Tap instances, the Signal Tap instance is part of the project and the exported partition. When you export a .qdb file, the file includes any Signal Tap HDL instances, unless you remove them, recompile the core, and re-export.

#### 7.5.1.2 Partition Boundary Ports

The core block Developer must identify and expose the potential Signal Tap points to the block Consumer. The Consumer uses the ports that you expose for Signal Tap debugging in the Consumer project.

##### 7.5.1.2.1 Debugging with the Synthesis Snapshot

If the .qdb you reuse is for the synthesized snapshot, adding pre-synthesis Signal Tap nodes is not possible, because that requires resynthesis of the core. However, you can add post-fit Signal Tap nodes, because the Fitter can connect and route the post-fit nodes.

To tap the post-fit nodes, the Consumer must:

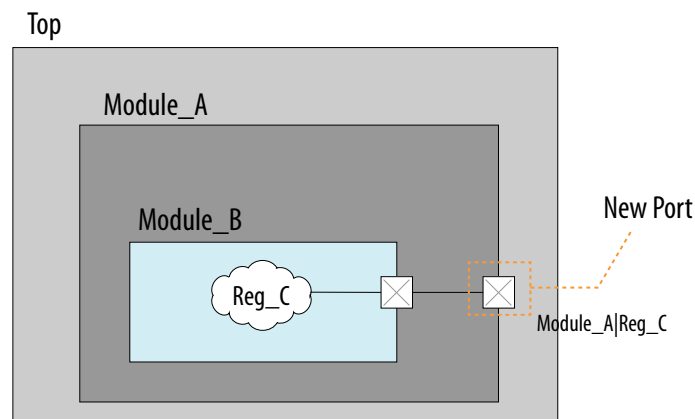
1. Add the root partition to a Consumer project, as [Step 4: Consumer: Add the Root Partition and Compile](#) on page 262 describes.
2. Compile the partition through the Fitter stage in the Consumer project.
3. Instantiate Signal Tap in the Consumer design and add the post-fit Signal Tap nodes.
4. To recompile the design from the Place stage, click **Processing > Start > Start Fitter (Place)**. The Fitter attaches the Signal Tap nodes to the existing synthesized nodes.

### 7.5.1.2.2 Debugging with the Final Snapshot

If the partition you reuse contains the final snapshot, the Consumer project placement and routing is complete, and adding new Signal Tap nodes is not possible. This limitation requires the partition Developer define additional boundary ports, before exporting the partition, that enable subsequent connection to the Signal Tap nodes. The partition Consumer then taps these ports for Signal Tap debugging in the Consumer project.

The Developer must create additional ports at the partition boundary, and connect the logic for probing to the new ports. Manual tunneling of connections through layers of RTL requires design changes, and can create issues with different versions of code. To avoid these complexities, use the **Create Boundary Partition Ports** assignment in the Assignment Editor to automatically create the additional ports and tunnel the logic without making RTL changes. The following figure shows two new boundary ports: `Module_A|Reg_C` and `Module_A|Module_B|Reg_C`

**Figure 109. Adding Ports at Partition Boundary**



The debug ports that the Developer creates with **Create Boundary Partition Ports** make a connection to the logic and tunnel the ports to the top-level partition. A new boundary port is then available at every child partition boundary. Any boundary ports you create become regular output ports on the exported partition.

### 7.5.1.2.3 Defining Partition Boundary Ports

Follow these steps to define partition boundary ports in a Developer project for subsequent Signal Tap debugging in a Consumer project.

1. Create the Developer project with the core partition for export.
2. Elaborate the design hierarchy.
3. To connect a design port to a partition boundary port, click **Assignments** > **Assignment Editor**, and then assign **Create Partition Boundary Ports** to one or more ports. When you assign a bus, the assignment applies to the root name of the debug port, with each bit enumerated. When the Consumer synthesizes the reused partition, all valid ports with the **Create Partition Boundary Ports** are visible in the Consumer project.
4. Compile the design. Following synthesis, view the partition boundary ports in the Create Partition Boundary Ports report. This report generates in the **In-System Debugging** folder under **Synthesis** reports.





5. To export the partition from the Developer project, click **Project** ► **Export Design Partition**.
6. From the original core partition source file, create a black box port definitions file that contains only port declarations. Add the partition boundary ports to the black box file.
7. Add the partition's .qdb, black box file, and any other data you require to the Consumer project.

#### 7.5.1.2.4 Connecting to Debug Ports in a Reused Partition

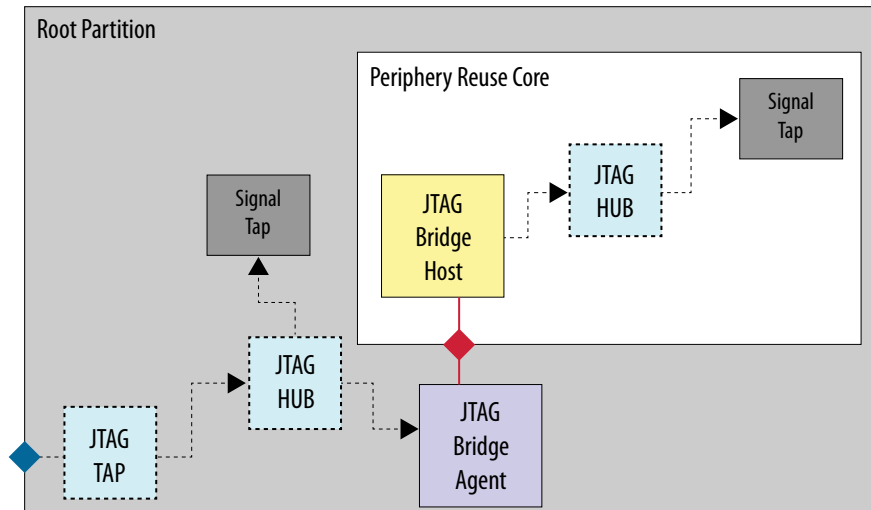
After defining partition boundary ports, follow these steps to use the debug ports from a .qdb in a Consumer project.

1. Define the boundary ports in the Developer project, as [Defining Partition Boundary Ports](#) on page 264 describes.
2. Create the Consumer project and add the black box file the Developer provides. The black box file must include the boundary ports from the Developer project. If the ports are not present in the ports list, the Consumer cannot add Signal Tap to the ports.
3. Elaborate the design hierarchy.
4. Click **Assignments** ► **Design Partitions Window**, and then define one or more partitions for the hierarchy branches that use the .qdb files. Specify the .qdb for the **Partition Database File** option for each partition that uses them, as [Creating and Modifying Design Partitions](#) on page 250 describes.
5. Click **File** ► **New** ► and then define a new Signal Tap Logic Analyzer file (.stp).
6. In the Signal Tap logic analyzer, use the Node Finder to locate the debug ports with the Signal Tap Pre-synthesis ports filter.
7. Add the clock source to your Signal Tap file.
8. Compile the design.
9. In the Compilation Report, under **Synthesis**, view the In-System Debugging report to verify the connection of the ports.
10. Debug your design in the Signal Tap logic analyzer.

#### 7.5.2 Signal Tap with Root Partition Reuse

Root partition reuse with Signal Tap requires a JTAG debug bridge to extend Signal Tap debugging from the root partition into the core partition. The Consumer of a root partition then uses Signal Tap to debug into the periphery reuse core partition by connecting to the debug bridge. The debug bridge allows independent debugging in the root and core partitions, using isolated Signal Tap logic in root and (optionally) core regions.

Figure 110. Signal Tap with JTAG Debug Bridge



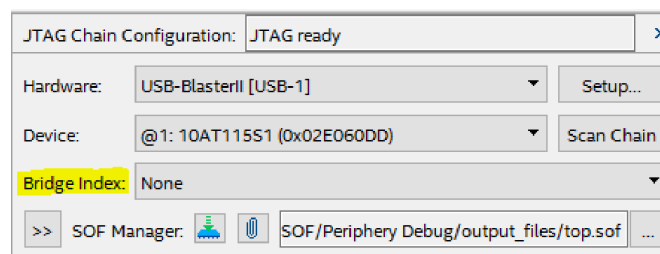
In root partition reuse, treat each partition individually at the time of integration. Each instance of Signal Tap can only connect within the partition that the instance resides. Therefore, the root partition and periphery reuse core partition each require separate Signal Tap files in this flow.

Although the root and core partitions share the same JTAG interface, use separate Signal Tap files for each reused partition. Use the core partition Signal Tap file only to debug the core partition. Each Signal Tap instance operates independently, and you must use each instance separately.

When configuring the Signal Tap logic analyzer, for the root partition, set the **Bridge Index** value to **None** in the JTAG Chain Configuration window.

*Note:* You must connect the SLD JTAG Bridge Agent to an SLD JTAG Bridge Host, or the Compiler generates an error.

Figure 111. JTAG Chain Configuration Bridge Index



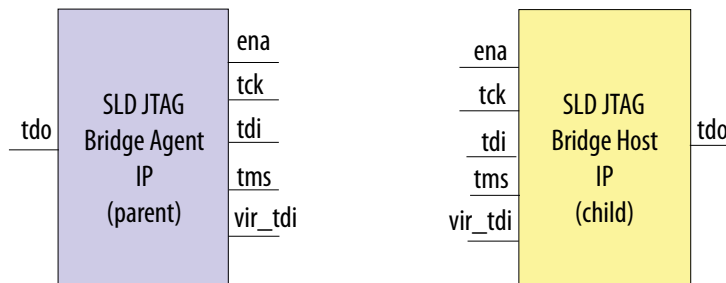
To directly debug the partition in the Developer project, the periphery Developer may include one or more Signal Tap and SLD JTAG Bridge instances in the Developer project. Then, any Signal Tap or SLD JTAG Bridge component the Developer instantiates in the periphery exports with the root partition, and is available in the Consumer project. If you do not want to expose Signal Tap points in the exported root partition, remove the Signal Tap file and recompile the Developer project prior to exporting the .qdb.



### 7.5.2.1 SLD JTAG Bridge Intel FPGA IP

The SLD JTAG Bridge Intel FPGA IP cores include the SLD JTAG Bridge Agent and SLD JTAG Bridge Host components. Instantiate one pair of SLD Bridge Agent and Host IP at the boundary of each periphery reuse core partition that requires debug.

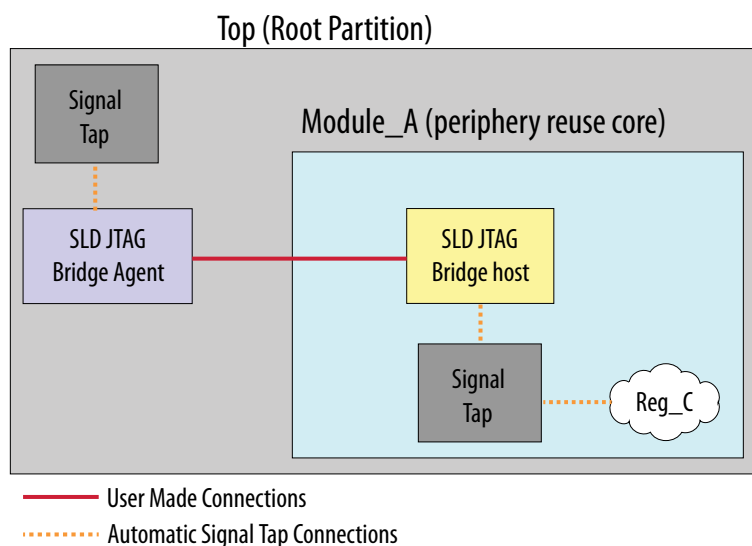
**Figure 112. SLD JTAG Bridge Agent and SLD Bridge Host IP Components**



- SLD JTAG Bridge Agent—enables debug of partial reconfiguration or periphery reuse core partitions, by extending the JTAG debug fabric from a higher-level partition to a partial reconfiguration or periphery reuse core partition containing the SLD JTAG Bridge Host. Instantiate the SLD JTAG Bridge Agent in the higher-level partition and connect the interface to an SLD JTAG Bridge Host in the child partition.
- SLD JTAG Bridge Host—enables debug of partial reconfiguration or periphery reuse core partitions by connecting the JTAG debug fabric in a partial reconfiguration or periphery reuse core partition to a higher-level partition containing the SLD JTAG Bridge Agent. Instantiate the SLD JTAG Bridge Host in the child partition and connect the interface to an SLD JTAG Bridge Agent in the higher-level partition.

When configuring the Signal Tap logic analyzer, for a periphery reuse core partition, set the bridge index according to the **Synthesis > In-System Debugging > JTAG Bridge Instance Agent Information** report in the Developer project.

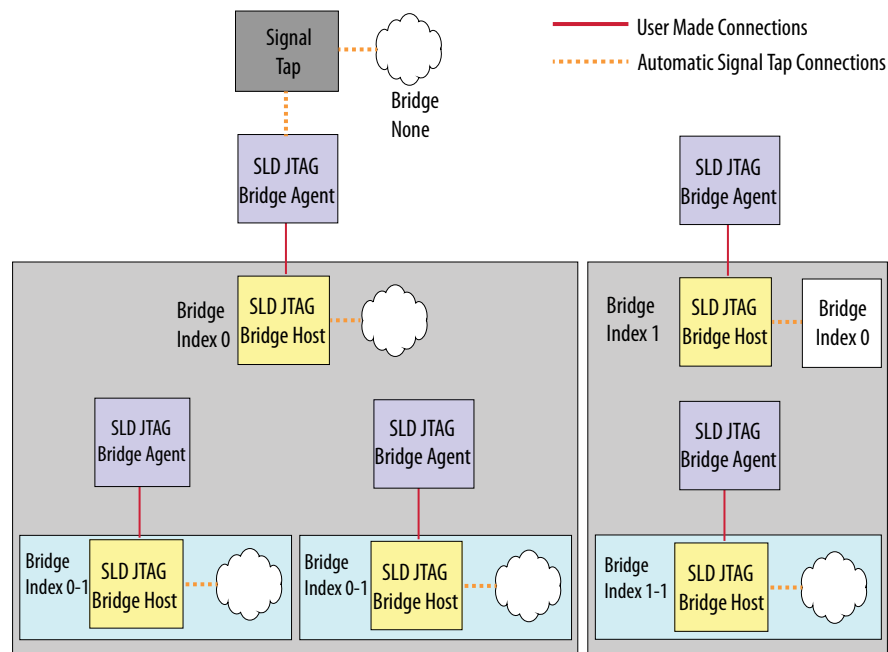
**Figure 113. SLD JTAG Bridge Component Configuration**



### 7.5.2.1.1 SLD JTAG Bridge Index

The Intel Quartus Prime software supports multiple instances of the SLD JTAG Bridge in partitions and their children. The Compiler assigns an index number to distinguish each instance. The bridge index for the root partition is always None. The Compilation Report lists the index numbers for the SLD JTAG Bridge Agents in the core partition.

Figure 114. SLD JTAG Bridge Index



*Note:* Following design synthesis, view the **Synthesis > In-System Debugging > JTAG Bridge Instance Agent Information** report in the Developer project. This report details how the bridge indexes are enumerated. The reports shows the hierarchy path and the associated index. The Developer must provide this information to the Consumer, so the Consumer understands the index mapping.

### 7.5.2.2 Instantiating the SLD JTAG Bridge in the Periphery Reuse Root Partition

Follow these steps to instantiate Signal Tap in the root partition:

1. Create an Intel Quartus Prime project that is setup for periphery reuse, as the steps in [Reusing Root Partitions](#) on page 259 describe.
2. From the IP Catalog, select, parameterize, and instantiate the SLD JTAG Bridge Agent IP core in the parent module of the core partition. Similarly, instantiate the SLD JTAG Bridge Host IP in the core partition. To display IP Catalog, click **Tools > IP Catalog**.



*Note:* You must connect the SLD JTAG Bridge Agent to an SLD JTAG Bridge Host, or the Compiler generates an error.

3. To export the root partition, click **Project** ► **Export Design Partition**. Specify the **Partition name** (root), the **Snapshot** level, and the .qdb file name for export. After the Developer creates the .qdb, the Consumer can reuse the exported .qdb file and optionally add Signal Tap to the core partition.
4. Compile and debug the Developer project. To run full compilation, click **Processing** ► **Start Compilation**.

#### Related Links

- [Debugging Designs with the Signal Tap Logic Analyzer](#)
- [Introduction to Intel FPGA IP Cores](#)
- [Reusing Root Partitions](#) on page 259

### 7.5.2.3 Instantiating Signal Tap in the Periphery Reuse Core Partition

Signal Tap debugging in the core partition requires the SLD JTAG Bridge IP in the root and core partitions. The SLD JTAG Bridge extends the JTAG connectivity into core partition in a Consumer project. The partition Consumer instantiates SLD JTAG Bridge Host to communicate with the SLD JTAG Bridge Agent that the partition Developer provides. After design synthesis, the Consumer can insert pre-synthesis Signal Tap points in the consumer project core partition.

1. In the New Project Wizard, create the Consumer project, assign the root partition with the **This project uses a Partition Database (.qdb) file for the root partition** setting, and then run **Analysis & Elaboration**.
2. In the core partition, instantiate the SLD JTAG Bridge Host IP core from the IP Catalog.
3. To run synthesis, click **Processing** ► **Start** ► **Start Analysis & Synthesis**.
4. Create a new Signal Tap file and add the pre-synthesis Signal Tap points from the core partition.

*Note:* Adding new Signal Tap points to the root partition is invalid. Any new Signal Tap points that you add to the root partition are unusable in the core partition's Signal Tap file.

5. To run full compilation, click **Processing** ► **Start Compilation**.

## 7.6 Creating a Top-Level Project for a Team-Based Design

In team-based designs that reuse design blocks, all contributors to the design ideally work within the same top-level project framework. Using the same project framework among team members ensures that all contributors have the correct settings and constraints that their partition requires.

This method helps to simplify timing closure when integrating the partitions into the top-level design. If some Developers do not have access to the top-level project framework, the team lead must communicate information about the project and constraints to those Developers.

The following steps describe preparing a top-level project that enables other Developers to provide optimized lower-level design partitions. The top-level project specifies the top-level entity, and then instantiates other design entities that other Developers optimize in a separate Intel Quartus Prime project.

1. Set up the top-level project and add source files, as normal. If the source files are incomplete, also create and add a black-box file to define the port directions for each incomplete module or entity, as [Step 3: Developer: Create a Black Box File](#) on page 258 describes.
2. Define design partitions for any instance that you want to maintain as a separate Intel Quartus Prime project, as [Creating and Modifying Design Partitions](#) on page 250 describes.
3. Define an empty partition for each design partition with unknown or incomplete definition, as [Defining an Empty Partition](#) on page 252 describes.
4. Create a Logic Lock region constraint for each partition that you maintain as a separate Intel Quartus Prime project. This physical partitioning of the device allows multiple team members to design independently without placement conflicts, as [Step 2: Developer: Define a Logic Lock Region](#) on page 260 describes.
5. To run full compilation, click **Processing > Start Compilation**.
6. Use one of the following methods to provide the top-level project information to design Developers:
  - If Developers have access to the top-level project framework, the team lead includes all settings and constraints the design requires. This framework includes clocks, PLLs, and other periphery interface logic that the Developer requires to optimize their partition. If Developers are part of the same design environment, they can check out a copy of the project files they require from the same source control system. This is the best method for sharing a set of project files. Otherwise, the team lead provides a copy of the top-level project (the design and corresponding .qsf assignments), so that each Developer creates their partition within the same project framework.
  - If a Developer does not have access to the top-level project framework, the team lead provides a Tcl script or other specifications to create a separate Intel Quartus Prime project that matches the top-level. The team lead also adds logic around the design block for export, so that the partition is consistent with the key characteristics of the top-level design environment. For example, the team lead can include a top-level PLL in the project, outside of the partition for export, so that Developers can optimize the design with information about the clocks and PLL parameters. This technique provides more accurate timing requirements. Export the partition for the top-level design, without exporting any auxiliary components that you instantiate outside the partition you are exporting.

### 7.6.1 Preparing a Lower-Level Partition for Integration

A Developer can follow these steps to prepare a lower-level design partition for integration with the top-level project:



1. Obtain a copy of the top-level project, or create a new project with the same assignments and constraints as the top-level project. Constrain the partition placement within a Logic Lock region, as [Step 2: Developer: Define a Logic Lock Region](#) on page 260 describes. Ensure that the design only uses the resources that the project lead allocates.
2. For each design partition that is not available or incomplete in the top-level project, set the **Empty** option to **Yes** in the Design Partitions Window. This setting creates an empty partition for later development.
3. For the low-level design partition under development, set the **Empty** option to **No**.
4. If the top-level project includes an empty wrapper stub file for the lower-level design partition, use that file as a template to create the partition logic, or replace the wrapper file with the appropriate source code that matches the same port definition.
5. When the lower level partition design is complete, follow the procedures in [Step 2: Developer: Compile and Export a Core Partition](#) on page 257. The project lead can now reuse the partition in the top-level project.

## 7.7 Document Revision History

This document has the following revision history.

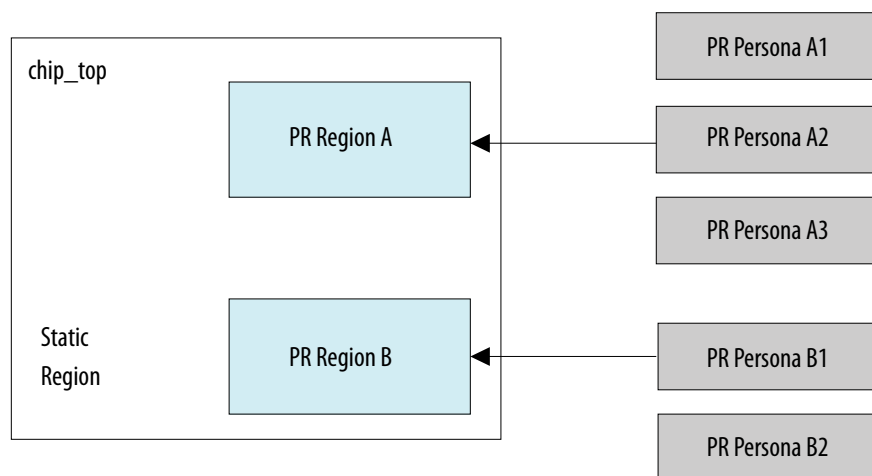
**Table 70. Document Revision History**

| Date       | Version | Changes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2017.11.06 | 17.1.0  | <ul style="list-style-type: none"> <li>• Reorganization of introduction and Incremental Block-Based Compilation.</li> <li>• Added <i>Design Partitioning</i> section.</li> <li>• Added <i>Debugging Block-Based Designs</i> section.</li> <li>• Added <i>Use Empty Partitions to Reduce Compilation Time</i> topic.</li> <li>• Removed requirement to add <code>.psmE</code>, <code>.msE</code>, and <code>.sof</code> to Consumer project.</li> <li>• Added Intel Stratix 10 support, including information about bundling of <code>.sdc</code> with exported partitions for Intel Stratix 10 designs.</li> <li>• Documented changes to Design Partitions window, Export Design Partition dialog box, and Logic Lock Regions window.</li> <li>• Added reference to new Design Partition Planner.</li> <li>• Updated references to corresponding <code>.qsE</code> assignments.</li> <li>• Changed references from periphery reuse to root partition reuse.</li> <li>• Rebranded for latest Intel standards.</li> </ul> |
| 2017.05.08 | 17.0.0  | <ul style="list-style-type: none"> <li>• First public release.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## 8 Creating a Partial Reconfiguration Design

Partial reconfiguration (PR) allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. You can define multiple personas for a particular region in your design, without impacting operation in areas outside this region. This methodology is effective in systems with multiple functions that time-share the same FPGA device resources. PR enables the implementation of more complex FPGA systems. The Intel Quartus Prime Pro Edition software supports the PR feature for the Intel Arria 10 and Intel Stratix 10 device families.

**Figure 115. Partial Reconfiguration Design**



PR provides the following advancements over a flat design:

- Allows run-time design reconfiguration
- Increases scalability of the design through time-multiplexing
- Lowers cost and power consumption through efficient use of board space
- Supports dynamic time-multiplexing functions in the design
- Improves initial programming time through smaller bitstreams
- Reduces system down-time through line upgrades
- Enables easy system update by allowing remote hardware change

Intel Quartus Prime Pro Edition software also supports hierarchical partial reconfiguration (HPR), with multiple parent and child design partitions, or multiple levels of partitions in a design. In HPR designs, a static region instantiates a parent PR region, and a parent PR region instantiates a child PR region. The same PR region reprogramming is possible for the child and parent partitions.





In addition, static update partial reconfiguration (SUPR) allows you to define and modify a specialized static region, without requiring recompilation of all personas. This technique is useful for a portion of a design that you may *possibly* want to change for risk mitigation, but that never requires runtime reconfiguration. In traditional PR, you must recompile all personas for any change to the static region. Refer to the Partial Reconfiguration Tutorials for complete information.

The Intel Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the PR control block for Intel Arria 10 designs. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition. Refer to [Partial Reconfiguration Simulation and Verification](#) on page 317 for complete PR simulation details.

### Related Links

- [Partial Reconfiguration Tutorials](#)
- [Intel Stratix 10 Device Overview](#)
- [Intel Stratix 10 Configuration User Guide](#)
- [Intel Arria 10 Device Overview](#)
- [Intel Arria 10 Configuration User Guide](#)
- [Intel Arria 10 Reconfiguration Interface and Dynamic Reconfiguration](#)

## 8.1 Partial Reconfiguration Basic Concepts

Implementing a PR design requires understanding the FPGA device capabilities and the Intel Quartus Prime IP components and compilation flow. The Intel Quartus Prime software includes the following IP cores that simplify PR implementation. You can instantiate one or more of these IP cores, or create your own PR handshake and freeze logic that interfaces with the PR region.

**Table 71. Partial Reconfiguration IP Cores**

| IP                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                              | Usage                                                      |
|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <b>Intel Arria 10 Partial Reconfiguration Controller</b>   | Dedicated IP component that sends the partial reconfiguration bitstream to the PR Control Block in the Intel Arria 10 FPGA. The PR bitstream performs reconfiguration by adjusting CRAM bits in the FPGA. Instantiate the IP in the static region of your design. This IP core interfaces with the PR control block to manage the bitstream source, and has a maximum clock frequency of 100MHz.         | One instance per FPGA, internal or external configuration. |
| <b>Intel Stratix 10 Partial Reconfiguration Controller</b> | Dedicated IP component that sends the partial reconfiguration bitstream to the Secure Device Manager in the Intel Stratix 10 FPGA. The PR bitstream performs reconfiguration by adjusting CRAM bits in the FPGA. Instantiate the IP in the static region of your design. The IP core interfaces with the Intel Stratix 10 FPGA secure device manager (SDM), and has a maximum clock frequency of 250MHz. | One instance per FPGA, internal configuration only.        |

*continued...*



| IP                                                     | Description                                                                                                                                                                                                   | Usage                                                        |
|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| <b>Partial Reconfiguration Region Controller</b>       | Provides a standard Avalon® Memory Mapped (Avalon-MM) interface to the block that controls handshaking with the PR region. Ensures that PR region stops, resets, and restarts, according to the PR handshake. | One instance per PR region.                                  |
| <b>Avalon-MM Partial Reconfiguration Freeze Bridge</b> | Provides freeze capabilities to the PR region for Avalon Memory Mapped (Avalon-MM) interfaces.                                                                                                                | One instance for each Avalon-MM interface in each PR region. |
| <b>Avalon-ST Partial Reconfiguration Freeze Bridge</b> | Provides freeze capabilities to the PR region for Avalon Streaming (Avalon-ST) interfaces.                                                                                                                    | One instance for each Avalon-ST interface in each PR region. |

**Table 72. Partial Reconfiguration Terminology**

| Term                                                       | Description                                                                                                                                                                                                                                                                                                             |
|------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Floorplan</b>                                           | The layout of physical resources on the device. Creating a design floorplan, or floorplanning, is the process of mapping logical design hierarchy to physical regions in the device.                                                                                                                                    |
| <b>Hierarchical Partial Reconfiguration</b>                | Partial reconfiguration that includes multiple parent and child design partitions, or nesting of partitions in the same design.                                                                                                                                                                                         |
| <b>PR control block</b> (Intel Arria 10 only)              | A dedicated block in the Intel Arria 10 FPGA. The PR control block processes the PR requests, handshake protocols, and verifies the cyclic redundancy check (CRC).                                                                                                                                                      |
| <b>PR host</b>                                             | The system for coordinating PR. The PR host communicates with the PR control block. Implement the PR host within the FPGA (internal PR host) or in a chip or microprocessor (external PR host for Intel Arria 10 designs only).                                                                                         |
| <b>PR partition</b>                                        | Design partition you designate for PR. A PR project can contain one or more partially reconfigurable PR partitions.                                                                                                                                                                                                     |
| <b>PR persona</b>                                          | A specific PR partition implementation in a PR region. A PR region can contain multiple personas. Static regions contain only one persona.                                                                                                                                                                              |
| <b>PR region</b>                                           | An area in the FPGA that you associate with a partially reconfigurable partition. A PR region contains the core locations of the device you want to reconfigure. A device can contain more than one PR region. A PR region can be core-only, such as LAB, RAM, or DSP.                                                  |
| <b>Revision</b>                                            | A collection of settings and constraints for one version of your project. An Intel Quartus Prime Settings File (.qsf) preserves each revision of your project. Your Intel Quartus Prime project can contain several revisions. Revisions allow you to organize several versions of your design within a single project. |
| <b>Secure Device Manager (SDM)</b> (Intel Stratix 10 only) | A triple-redundant processor-based Intel Stratix 10 FPGA block that performs authentication, decryption, and decompression on the configuration data the block receives, before sending the data over to the configurable nodes through the configuration network.                                                      |
| <b>Snapshot</b>                                            | The output of a Compiler stage. The Intel Quartus Prime Pro Edition Compiler generates a snapshot of the compiled database for each partition, after each compilation stage. Export the snapshot at various stages of the compilation flow, such as synthesis or final.                                                 |
| <b>Static region</b>                                       | All areas outside the PR regions in your project. You associate the static region with the top-level partition of the design. The static region contains both the core and periphery locations of the device.                                                                                                           |
| <b>Static update partial reconfiguration</b>               | A specialized static region that allows change, without requiring the recompilation of all personas. This technique is useful for a portion of a design that you may <i>possibly</i> want to change for risk mitigation, but that never requires runtime reconfiguration.                                               |

### Related Links

[Partial Reconfiguration IP Solutions User Guide](#)

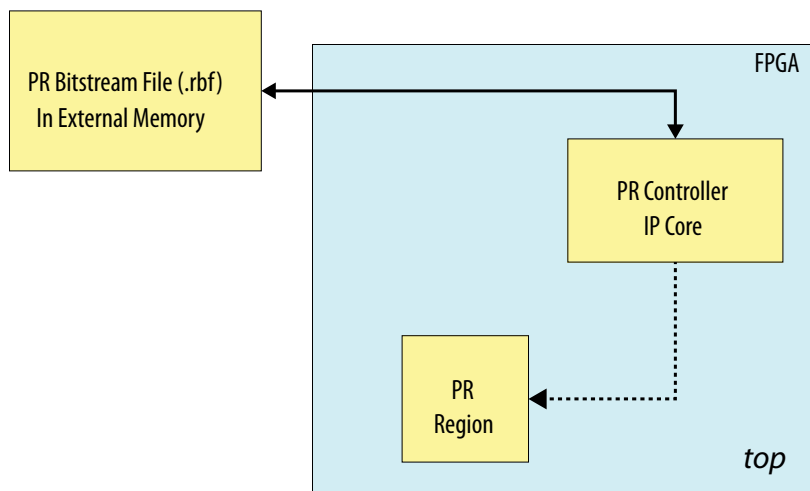
## 8.2 Internal Host Partial Reconfiguration

In internal host control, an internal controller, a Nios II processor, or an interface such as PCI Express (PCIe) or ethernet, communicates directly with the Intel Arria 10 PR control block, or with the Intel Stratix 10 SDM.

To transfer the PR bitstream into the PR control block or SDM, use the Avalon-MM interface on the Intel Arria 10 or Intel Stratix 10 Partial Reconfiguration IP core. When the device enters user mode, initiate partial reconfiguration through the FPGA core fabric using the PR internal host.

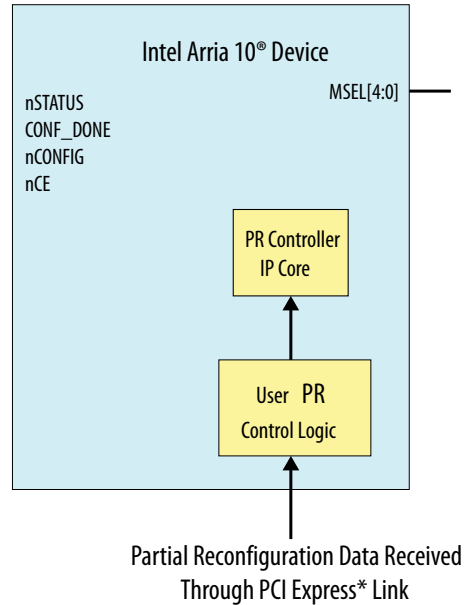
*Note:* If you create your own control logic for the PR host, the logic must meet the PR interface requirements.

**Figure 116. Internal Host PR**



When performing partial reconfiguration with an internal host, use the dedicated PR pins (`PR_REQUEST`, `PR_READY`, `PR_DONE`, and `PR_ERROR`) as regular I/Os. Implement your static region logic to retrieve the PR programming bitstreams from an external memory, for processing by the internal host.

**Figure 117. Intel Arria 10 FPGA System Using an Internal PR Host**

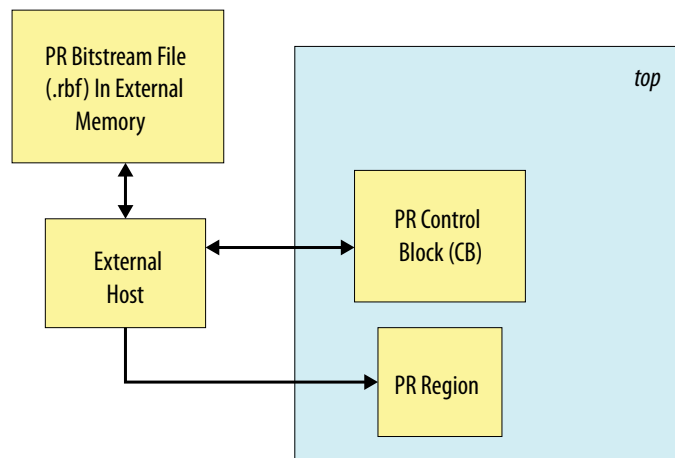


Send the programming bitstreams for partial reconfiguration through the PCI Express link. Then, you process the bitstreams with your PR control logic and send the bitstreams to the PR IP core for programming.

### 8.3 External Host Partial Reconfiguration (Intel Arria 10 Designs Only)

In external host control, an external FPGA or CPU controls the PR configuration using external dedicated PR pins on the target device. The current version of the Intel Quartus Prime Pro Edition software supports external host PR configuration only for Intel Arria 10 devices. When using an external host, you must implement the control logic for sending the bitstream to the hard FPGA programming pins.

**Figure 118. PR System Using an External Host**



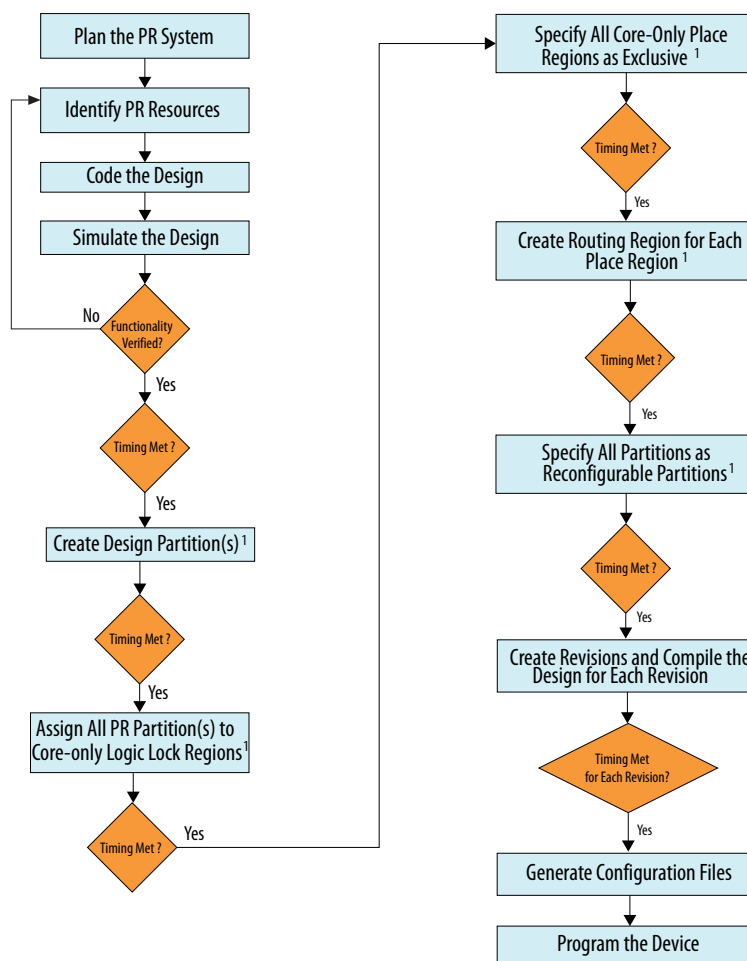


## 8.4 Partial Reconfiguration Design Flow

The PR design flow requires initial planning. This planning involves setting up one or more design partitions, and then determining the placement assignments in the floorplan. Well-planned PR partitions improve design area utilization and performance. The Intel Quartus Prime software also allows you to create nested PR regions as part of an HPR flow. Reprogramming a child PR region does not affect the parent or the static region. In the HPR flow, reprogramming the parent region, reprograms the associated child region with the default child persona, without affecting the static region. The HPR flow does not impose any restrictions on the number of sub-partitions you can create in your design.

The PR design flow uses the project revisions feature in the Intel Quartus Prime software. Your initial design is the base revision, where you define the static region boundaries and reconfigurable regions on the FPGA. From the base revision, you create multiple revisions. These revisions contain the different implementations for the PR regions. However, all PR implementation revisions use the same top-level placement and routing results from the base revision.

**Figure 119. Partial Reconfiguration Design Flow**





To debug your design at a later stage using the Signal Tap Logic Analyzer, you must instantiate the SLD JTAG bridge Intel FPGA IP for each PR region in your design. The SLD JTAG Bridge consists of two IP components - SLD JTAG Bridge Agent and SLD JTAG Bridge Host. Perform the following steps during the early planning stage, to ensure you can signal tap your static as well as PR region, at a later stage:

1. Instantiate the SLD JTAG Bridge Agent IP in the static region.
2. Instantiate the SLD JTAG Bridge Host IP in the PR region of the default persona.
3. Then, instantiate the SLD JTAG Bridge Host IP for each of the personas during the synthesis revision creation for the personas.

For more information about the SLD JTAG bridge instantiation, refer to *Instantiating a SLD JTAG Bridge Agent* and *Instantiating a SLD JTAG Bridge Host* sections in the Intel Quartus Prime Pro Edition handbook.

**Note:** The Intel Quartus Prime software does not support the use of PR in combination with any other hierarchical design flow.

#### Related Links

- [Instantiating a SLD JTAG Bridge Agent](#)
- [Instantiating a SLD JTAG Bridge Host](#)

### 8.4.1 Identifying Partial Reconfiguration Resources

When designing for partial reconfiguration, you must first determine the logical hierarchy boundaries that you can define as reconfigurable partitions. Next, set up the design hierarchy and source code to support this partitioning.

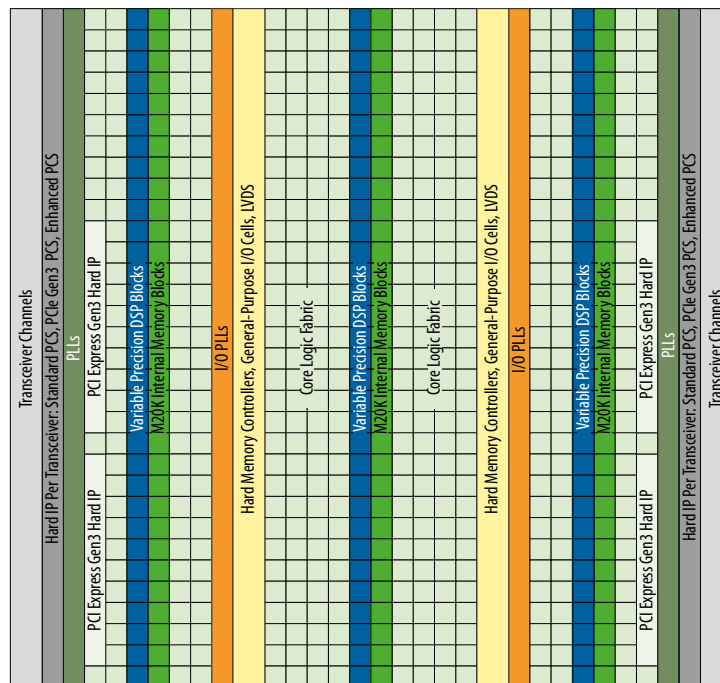
Reconfigurable partitions can contain only core resources, such as LABs, embedded memory blocks (M20Ks and MLABs), and DSP blocks in the FPGA. All periphery resources, such as transceivers, external memory interfaces, GPIOs, I/O receivers, and hard processor system (HPS), must be in the static portion of the design. Partial reconfiguration of global network buffers for clocks and resets is not possible.

**Table 73. Supported Reconfiguration Methods in Intel Arria 10 and Intel Stratix 10 Devices**

| Hardware Resource Block   | Reconfiguration Method  |
|---------------------------|-------------------------|
| Logic Block               | Partial reconfiguration |
| Digital Signal Processing | Partial reconfiguration |
| Memory Block              | Partial reconfiguration |
| Core Routing              | Partial reconfiguration |
| Transceivers              | Dynamic reconfiguration |
| PLL                       | Dynamic reconfiguration |
| I/O Blocks                | Not supported           |
| Clock Control Blocks      | Not supported           |



Figure 120. Available Resource Types in Intel Arria 10 Devices



Use any Intel Quartus Prime-supported design entry method to create core-only logic for a PR partition, including Platform Designer, the Intel HLS Compiler, or standard SystemVerilog, Verilog HDL, and VHDL design files.

The following Intel FPGA IP cores support system-level debugging in the static region:

- In-System Memory Content Editor
- In-System Sources and Probes Editor
- Virtual JTAG
- Nios II JTAG Debug Module
- Signal Tap Logic Analyzer

**Note:** Only Signal Tap Logic Analyzer allows simultaneous debugging of the static and PR regions.

### 8.4.2 Defining PR Partitions

Create design partitions for each PR region that you want to partially reconfigure. You can create any number of independent partitions or PR regions in your design. Create design partitions for partial reconfiguration from the Project Navigator, or the Design Partitions Window.

A design partition is the logical partitioning of the design, and does not specify a physical area on the device. Associate the partition with a specific area of the FPGA using Logic Lock Region floorplan assignments. To avoid partitions obstructing design optimization, group the logic together within the same partition. If your design includes a hierarchical PR flow with parent and child partitions, you can assign multiple parent or child partitions to your design, as well as multiple levels of PR partitions.

Standard hierarchical design practices help achieve successful partial reconfiguration. Follow these guidelines when creating partitions for PR regions in your design:

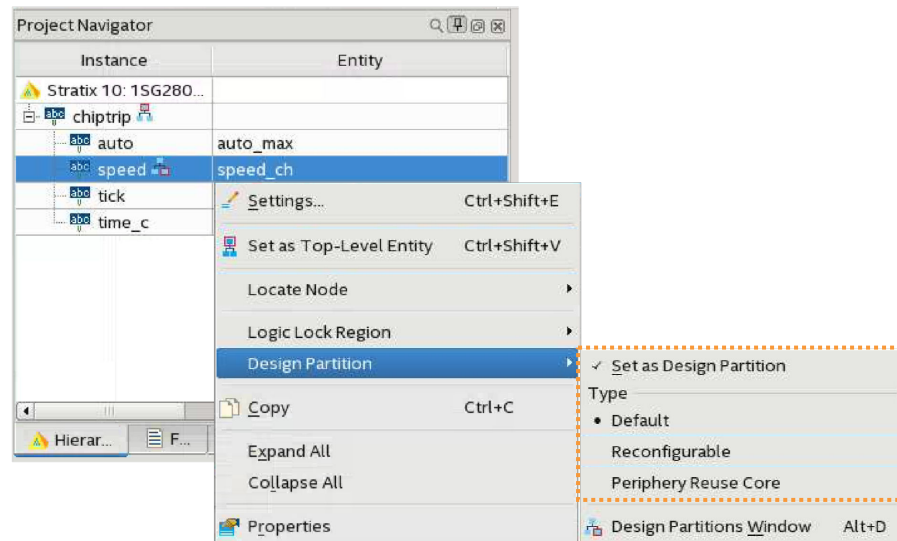
- Register all the partition boundaries, and all the inputs and outputs of each partition.
- Minimize the number of paths crossing the partition boundaries.
- Minimize the timing-critical paths passing in or out of the PR regions. In case of timing critical-paths crossing the PR region boundaries, rework the PR regions to avoid these paths.
- Avoid creating reset or clock signals inside the PR regions.

Follow these steps to create and modify design partitions:

### Creating a Partition from the Project Navigator

1. Click **Processing** ► **Start** ► **Start Analysis & Elaboration**.
2. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition** ► **Set as Design Partition**. A design partition icon appears next to each instance that is set as a partition.

**Figure 121. Creating Design Partitions from Project Navigator**



3. To define the partition **Type**, right-click the instance in the **Hierarchy** tab, click **Design Partition** ► **Reconfigurable**. You can only define the partition **Type** after setting the instance as a partition.

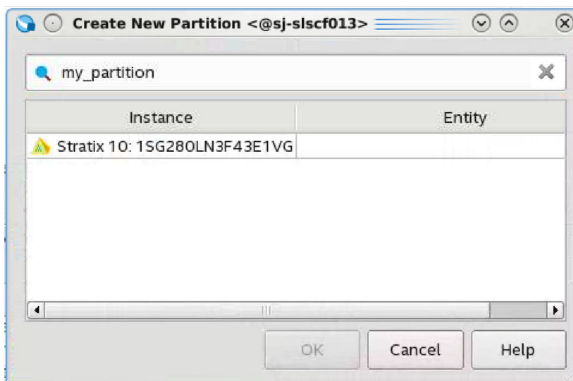
### Defining Partitions in the Design Partitions Window

1. To view and edit all design partitions in the project, click **Assignments** ► **Design Partitions Window**.
2. To define a new partition, double-click the **<<new>>** button in the **Partition Name** column.
3. Select the design instance to partition and click **OK**.



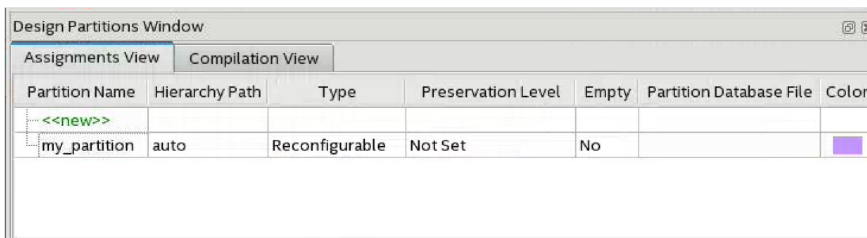


Figure 122. Create New Partition Window



4. For PR partitions, double-click the **Type** column and select the **Reconfigurable** partition type.

Figure 123. Design Partitions Window



The **Color** column indicates the color of each partition. The Design Partition Planner and Chip Planner also display this color for the partition. Right-click a partition in the window to perform various tasks, such as deleting the partition, locating the node, or creating Logic Lock regions for the partition.

The Intel Quartus Prime software automatically generates a partition name, based on the instance name and hierarchy path. This default partition name varies with each instance. Edit the partition name in the Design Partitions Window by double-clicking the name.

The following assignments in the .qsf file correspond to the design partition creation in the Design Partitions Window:

```
set_instance_assignment -name PARTITION pr_partition -to <design_instance>
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to /
<design_instance>
```

### 8.4.3 Defining Personas

Your partial reconfiguration design can have multiple PR partitions, each with multiple personas. Each of these personas function differently. However, all the PR personas must use the same set of signals to interact with the static region. Ensure that the signals interacting with the static region are a super-set of all the signals in all the personas. A PR design requires an identical I/O interface for each persona in the PR region.

### 8.4.3.1 Creating Wrapper Logic for PR Regions

If all personas for your design do not have identical top-level interfaces, you must create the wrapper logic to ensure that all the personas appear similar to the static region. Define a wrapper for each persona, and instantiate the persona logic within the wrapper. If all personas have identical top-level interfaces, the personas do not require wrapper logic. In this wrapper, you can create dummy ports to ensure that all the personas of a PR region have the same connection to the static region.

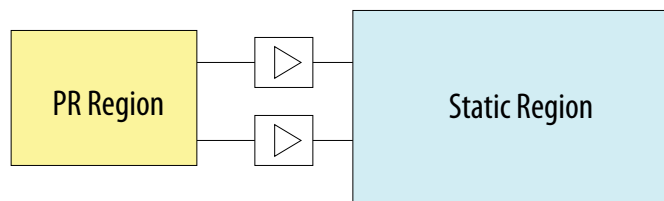
During the PR compilation, the Compiler converts each of the non-global ports on interfaces of the PR region into boundary port wire LUTs. The naming convention for boundary port wire LUTs are `<input_port>~IPORT` for input ports, and `<output_port>~OPORT` for output ports. For example, the instance name of the wire LUT for an input port with the name `my_input`, on a PR region with the name `my_region`, is `my_region|my_input~IPORT`.

1. Manually floorplan the boundary ports using the Logic Lock region assignments, or place the boundary ports automatically using the Fitter. The Fitter places the boundary ports during the base revision compile. The boundary LUTs are invariant locations the Fitter derives from the persona you compile. These LUTs represent the boundaries between the static region and the PR routing and logic. The placement remains stationary regardless of the underlying persona, because the routing from the static logic does not vary with a different persona implementation.
2. To constrain all boundary ports within a given region, use a wildcard assignment. For example:

```
set_instance_assignment -name PLACE_REGION "65 59 65 85" -to \
    u_my_top|design_inst|pr_inst|pr_inputs.data_in*~IPORT
```

This assignment constrains all the wire LUTs corresponding to the IPORTS that you specify within the place region, between the coordinates (65 59) and (65 85).

**Figure 124. Wire-LUTs at the PR Region Boundary**



Optionally, floorplan the boundary ports down to the LAB level, or individual LUT level. To floorplan to the LAB level, create a 1x1 Logic Lock `PLACE_REGION` constraint (single LAB tall and a single LAB wide). Optionally, specify a range constraint by creating a Logic Lock placement region that spans the range. For more information about floorplan assignments, refer to *Floorplan the Partial Reconfiguration Design*.

#### Related Links

[Floorplanning a Partial Reconfiguration Design](#) on page 295  
For more information on floorplanning your design.

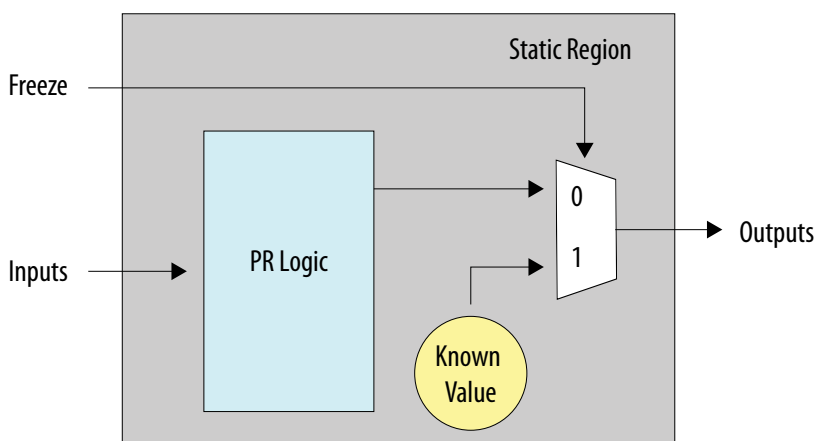
### 8.4.3.1.1 Freeze Logic for PR Regions

When partially reconfiguring a design, freeze all the outputs of each PR region to a known constant value. This freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process.

The PR region cannot drive valid data until the partial reconfiguration process is complete, and the PR region is reset. Freezing is important for control signals that you drive from the PR region.

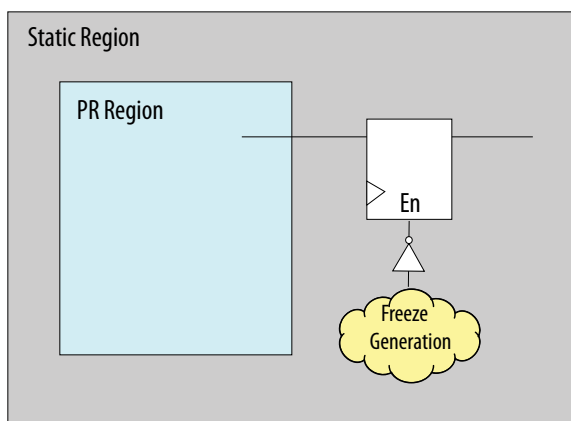
The freeze technique that you choose is optional, depending on the particular characteristics of your design. The freeze logic must reside in the static region of your design. A common freeze technique is to instantiate 2-to-1 multiplexers on each output of the PR region, to hold the output constant during partial reconfiguration.

**Figure 125. Freeze Technique #1 for Intel Arria 10 Devices**



An alternative freeze technique is to register all outputs of the PR region in the static region. Then, use an enable signal to hold the output of these registers constant during partial reconfiguration.

**Figure 126. Freeze Technique #2 for Intel Arria 10 Devices**



The Partial Reconfiguration Region Controller IP core includes a freeze port for the region that it controls. Include this IP component with your system-level control logic to freeze the PR region output. For designs with multiple PR regions, instantiate one

PR Region Controller IP core for each PR region in the design. The Intel Quartus Prime software includes the Avalon-MM Freeze Bridge and Avalon-ST Freeze Bridge Intel FPGA IP cores. You can use these IP cores to implement freeze logic, or design your own freeze logic.

The static region logic must be independent of all the outputs from the PR regions for a continuous operation. Control the outputs of the PR regions by adding the appropriate freeze logic for your design.

**Note:** There is no requirement to freeze the global and non-global inputs of a PR region for Intel Arria 10 or Intel Stratix 10 devices.

**Related Links**

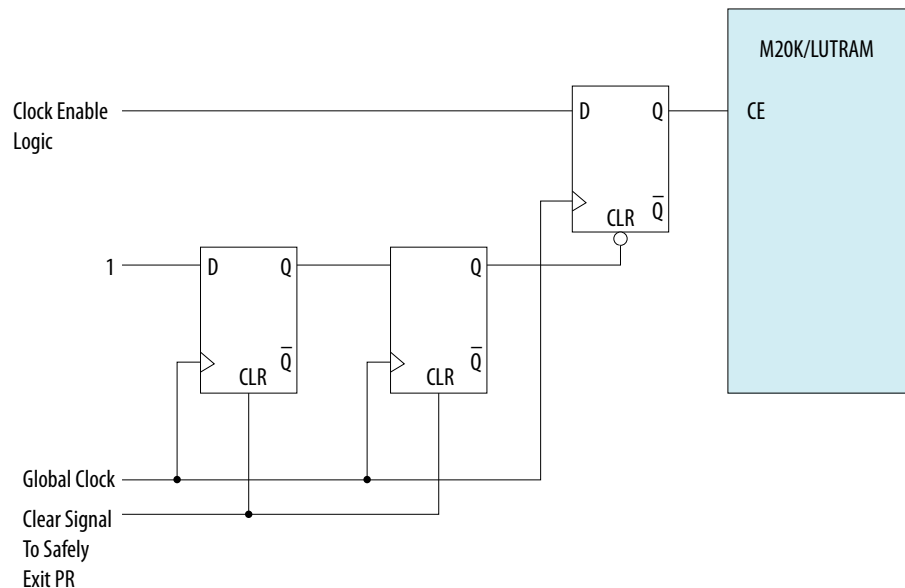
[Partial Reconfiguration IP Solutions User Guide](#)

**8.4.3.2 Implementing Clock Enable for On-Chip Memories with Initialized Contents**

Follow these guidelines to implement clock enable for on-chip memories with initialized contents:

1. To avoid spurious writes during PR programming for memories with initialized contents, implement the clock enable circuit in the same PR region as the M20K or MLAB RAM. This circuit depends on an active-high clear signal from the static region.
2. Before you begin the PR programming, assert this signal to disable the memory's clock enable. Your system PR controller must deassert the clear signal on PR programming completion. You can use the freeze signal for this purpose.
3. Use the Intel Quartus Prime IP Catalog or Platform Designer to instantiate the On-Chip Memory and RAM Intel FPGA IP cores that include an option to automatically add this circuitry.

**Figure 127. RAM Clock Enable Circuit for PR Region**





### Example 70. Verilog RTL for Clock Enable

```

reg ce_reg;
reg [1:0] ce_delay;

always @(posedge clock, posedge freeze) begin
  if (freeze) begin
    ce_delay <= 2'b0;
  end
  else begin
    ce_delay <= {ce_delay[0], 1'b1};
  end
end

always @(posedge clock, negedge ce_delay[1]) begin
  if (~ce_delay[1]) begin
    ce_reg <= 1'b0;
  end
  else begin
    ce_reg <= clken_in;
  end
end

wire ram_wrclocken;
assign ram_wrclocken = ce_reg;

```

#### Related Links

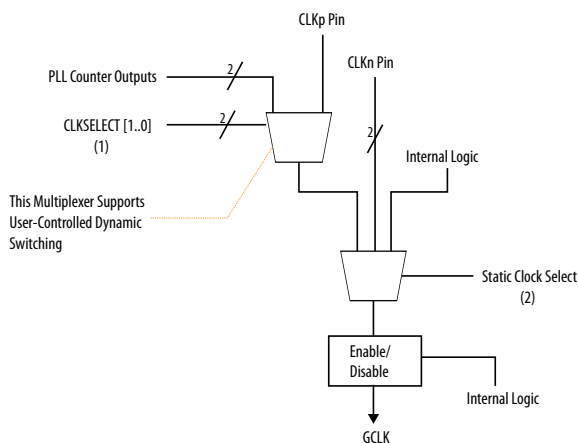
[Embedded Memory User Guide](#)

#### 8.4.3.2.1 Clock Gating

An alternate method to avoid spurious writes of initialized content memories is to gate the clock feeding the memories in the static region of your design. Clock gating is logically equivalent to using clock enable on the memories. This method provides the following features:

- Uses the enable port of the global clock buffers to disable the clock before starting the partial reconfiguration operation. Also enables the clock on PR completion.
- Ensures that the clock does not switch during reconfiguration, and requires no additional logic to avoid spurious writes.

Figure 128. Global Clock Control Block



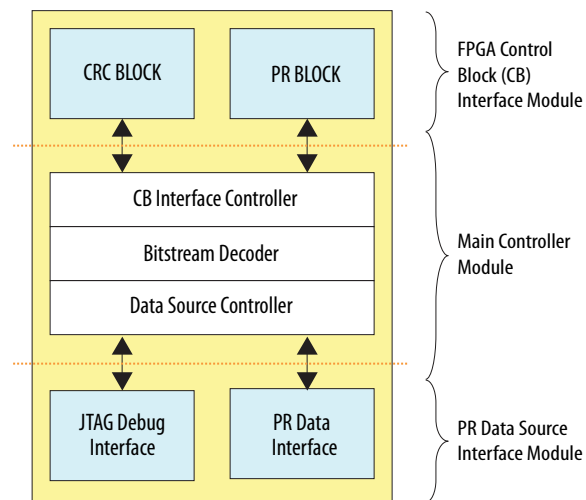
**Related Links**

[Clock Control Block \(ALTCLKCTRL\) Intel FPGA IP User Guide](#)

**8.4.4 Instantiating the Intel Arria 10 PR Controller IP**

When you instantiate the Intel Arria 10 Partial Reconfiguration Controller IP core from the IP Catalog (**Tools > IP Catalog**), the Intel Quartus Prime software automatically connects the IP core to the PR control block. If you create your own custom logic to perform the function of the IP core, manually instantiate the control block to communicate with the FPGA system.

**Figure 129. Intel Arria 10 Partial Reconfiguration Controller**



The Intel Arria 10 Partial Reconfiguration Controller IP core interfaces with the PR control block to manage the bitstream source. Use this IP core in an Intel Arria 10 design when performing partial reconfiguration using an internal PR host, Nios II, PCI Express, or Ethernet. Instantiate the IP core from the Intel Quartus Prime IP Catalog or Platform Designer.

During partial reconfiguration, send a PR bitstream stored outside the FPGA to the PR control block inside the FPGA. This communication enables the control block to update the CRAM bits necessary for configuring the PR region in the FPGA. The PR bitstream contains the instructions (opcodes) and the configuration bits necessary for reconfiguring a specific PR region.

**Related Links**

[Partial Reconfiguration Solutions IP User Guide](#)

**8.4.4.1 PR Control Block and CRC Block VHDL Component Declaration (Intel Arria 10 Designs Only)**

To manually instantiate the PR control block and the CRC block in your Intel Arria 10 PR design:



1. Use the code sample below, containing the component declaration in VHDL. This code performs the PR function from within the core (code block within Core\_Top).

```
module Chip_Top is port (  
    --User I/O signals (excluding signals that relate to PR)  
    ..  
    ..  
)  
-- Following shows the connectivity within the Chip_Top module  
Core_Top : Core_Top  
port_map (  
    ..  
    ..  
);  
  
m_pr : twentynm_prblock  
port map(  
    clk => dclk,  
    correct1 => '1', --1 - when using PR from inside  
    --0 - for PR from pins; You must also enable  
    -- the appropriate option in Quartus Prime settings  
    prequest => pr_request,  
    data => pr_data,  
    error => pr_error,  
    ready => pr_ready,  
    done => pr_done  
);  
  
m_crc : twentynm_crcblock  
port map(  
    shiftnld => '1', --If you want to read the EMR register when  
    clk => dummy_clk, --error occurs, refer to AN539 for the  
    --connectivity for this signal. If you only want  
    --to detect CRC errors, but plan to take no  
    --further action, you can tie the shiftnld  
    --signal to logical high.  
    crcerror => crc_error  
);
```

*Note:* This VHDL example is adaptable for Verilog HDL instantiation.

2. Add additional ports to Core\_Top to connect to both components.
3. Follow these rules when connecting the PR control block to the rest of your design:
  - Set the `correct1` signal to '1' (when using partial reconfiguration from core) or to '0' (when using partial reconfiguration from pins).
  - The `correct1` signal must match the **Enable PR pins** option setting in the **Device and Pin Options** dialog box (**Assignments > Device > Device and Pin Options**).
  - When performing partial reconfiguration from pins, the Fitter automatically assigns the PR unassigned pins. Assign all the dedicated PR pins using Pin Planner (**Assignments > Pin Planner**) or Assignment Editor (**Assignments > Assignment Editor**).
  - When performing partial reconfiguration from the core logic, connect the `prblock` signals to either core logic or I/O pins, excluding the dedicated programming pin, such as DCLK.

#### 8.4.4.1.1 PR Control Block and CRC Block VHDL Instantiation (Intel Arria 10 Designs Only)

The following example instantiates a PR control block inside your top-level Intel Arria 10 project, `Chip_Top`, in VHDL:

```

module Chip_Top is port (
  --User I/O signals (excluding signals that relate to PR)
  ..
  ..
)
-- Following shows the connectivity within the Chip_Top module
Core_Top : Core_Top
  port_map (
    ..
    ..
  );
m_pr : twentynm_prblock
  port map(
    clk => dclk,
    corectl => '1', --1 - when using PR from inside
    --0 - for PR from pins; You must also enable
    -- the appropriate option in Quartus Prime settings
    prequest => pr_request,
    data => pr_data,
    error => pr_error,
    ready => pr_ready,
    done => pr_done
  );
m_crc : twentynm_crcblock
  port map(
    shiftnld => '1', --If you want to read the EMR register when
    clk => dummy_clk, --error occurs, refer to AN539 for the
    --connectivity for this signal. If you only want
    --to detect CRC errors, but plan to take no
    --further action, you can tie the shiftnld
    --signal to logical high.
    crcerror => crc_error
  );

```

#### 8.4.4.1.2 PR Control Block and CRC Block Verilog HDL Instantiation (Intel Arria 10 Designs Only)

The following example instantiates a PR control block inside your top-level Intel Arria 10 PR project, `Chip_Top`, in Verilog HDL:

```

Chip_Top:
module Chip_Top (
  //User I/O signals (excluding PR related signals)
  ..
  ..
  //PR interface and configuration signals declaration
  wire pr_request;
  wire pr_ready;
  wire pr_done;
  wire crc_error;
  wire dclk;
  wire [31:0] pr_data;

  twentynm_prblock m_pr
  (
    .clk (dclk),
    .corectl (1'b1),
    .prequest(pr_request),
    .data (pr_data),
    .error (pr_error),
    .ready (pr_ready),
    .done (pr_done)
  )

```





```
);
twentynm_crcblock m_crc
(
  .clk (clk),
  .shiftnld (1'b1),
  .crcerror (crc_error)
);
endmodule
```

For more information about port connectivity for reading the Error Message Register (EMR), refer to the *AN539: Test Methodology of Error Detection and Recovery using CRC*.

### Related Links

[AN539: Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices](#)

## 8.4.4.2 Partial Reconfiguration Control Block Signals (Intel Arria 10 Designs Only)

The following table lists the partial reconfiguration control block interface signals:

**Table 74. PR Control Block Interface Signals**

| Signal     | Width  | Direction | Description                                                                                    |
|------------|--------|-----------|------------------------------------------------------------------------------------------------|
| pr_data    | [31:0] | Input     | Carries the configuration bitstream.                                                           |
| pr_done    | 1      | Output    | Indicates that the PR process is complete.                                                     |
| pr_ready   | 1      | Output    | Indicates that the control block is ready to accept PR data from the control logic.            |
| pr_error   | 1      | Output    | Indicates a partial reconfiguration error.                                                     |
| pr_request | 1      | Input     | Indicates that the PR process is ready to begin.                                               |
| corectl    | 1      | Input     | Determines whether you are performing the partial reconfiguration internally, or through pins. |

**Note:**

- Use data signal width of x8, x16, or x32 in your PR design.
- All the inputs and outputs are asynchronous to the PR clock (`clk`), except data signal. `data` signal is synchronous to `clk` signal.
- PR clock must be free-running.
- `data` signal must be 0 while waiting for `ready` signal.

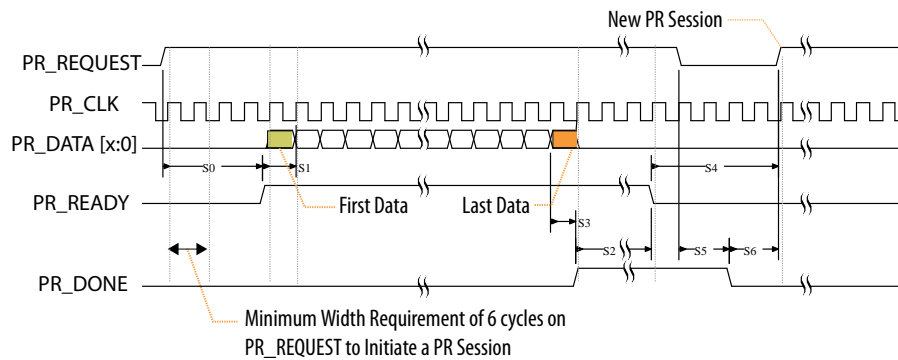
### 8.4.4.2.1 PR Control Block Signals Timing Diagrams (Intel Arria 10 Designs Only)

#### Successful PR Session (Intel Arria 10 Example)

The following flow describes a successful Intel Arria 10 PR session:

1. Assert `PR_REQUEST` and wait for `PR_READY`; drive `PR_DATA` to 0.
2. The PR control block asserts `PR_READY`, asynchronous to `clk`.
3. Start sending Raw Binary File (`.rbf`) to the PR control block, with 1 valid word per clock cycle. On `.rbf` file transfer completion, drive `PR_DATA` to 0. The PR control block asynchronously asserts `PR_DONE` when the control block completes the reconfiguration operation. The PR control block deasserts `PR_READY` on configuration completion.
4. Deassert `PR_REQUEST`. The PR control block acknowledges the end of `PR_REQUEST`, and deasserts `PR_DONE`. The host can now initiate another PR session.

**Figure 130. Timing Diagram for Successful Intel Arria 10 PR Session**



#### Related Links

[Generating Raw Binary Programming Files](#) on page 316

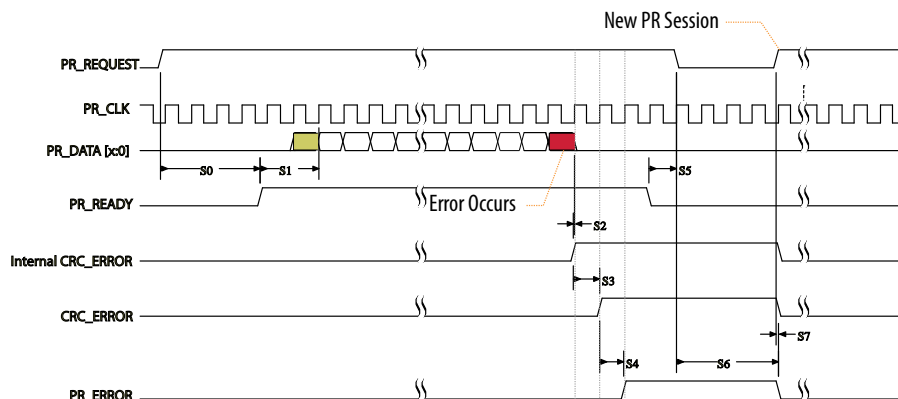
#### Unsuccessful PR Session with Configuration Frame Readback Error (Intel Arria 10 Example)

The following flow describes an Intel Arria 10 PR session with error in the EDCRC verification of a configuration frame readback:

1. The PR control block internally detects a CRC error.
2. The CRC control block then asserts `CRC_ERROR`.
3. The PR control block asserts the `PR_ERROR`.
4. The PR control block deasserts `PR_READY`, so that the host can withdraw the `PR_REQUEST`.
5. The PR control block deasserts `CRC_ERROR` and clears the internal `CRC_ERROR` signal to get ready for a new PR session. The host can now initiate another PR session.



**Figure 131. Timing Diagram for Unsuccessful Intel Arria 10 PR Session with Configuration Frame Readback Error**

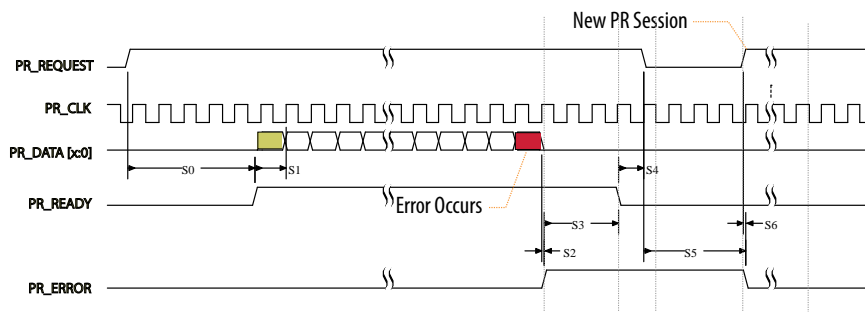


**Unsuccessful PR Session with PR\_ERROR (Intel Arria 10 Example)**

The following flow describes an Intel Arria 10 PR session with transmission error or configuration CRC error:

1. The PR control block asserts PR\_ERROR.
2. The PR control block deasserts PR\_READY, so that the host can withdraw PR\_REQUEST.
3. The PR control block deasserts PR\_ERROR to get ready for a new PR session. The host can now initiate another PR session.

**Figure 132. Timing Diagram for Unsuccessful Intel Arria 10 PR Session with PR\_ERROR**

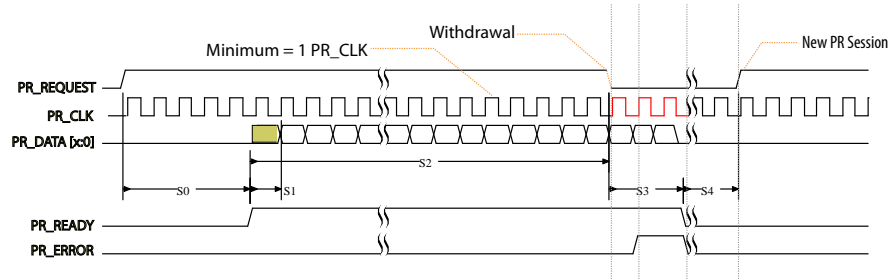


**Late Withdrawal PR Session (Intel Arria 10 Example)**

The following flow describes a late withdrawal Intel Arria 10 PR session:

1. The PR host can withdraw the request after the PR control block asserts PR\_READY.
2. The PR control block deasserts PR\_READY. The host can now initiate another PR session.

Figure 133. Timing Diagram for Late Withdrawal Intel Arria 10 PR Session

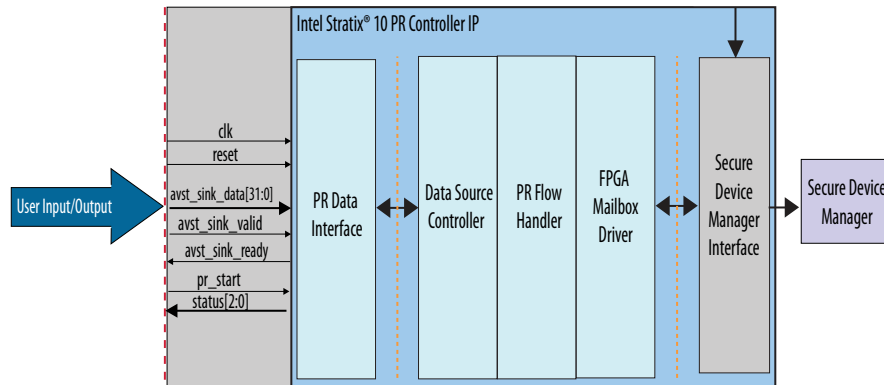


**Note:** The PR host can withdraw the request any time before the controller asserts PR\_READY. Therefore, the PR host must not return until the PR control block asserts PR\_READY. Provide at least 10 PR\_CLK cycles after deassertion of PR\_REQUEST, before requesting a new PR session.

### 8.4.5 Instantiating the Intel Stratix 10 PR Controller IP

When you instantiate the Intel Stratix 10 Partial Reconfiguration Controller IP core from the IP catalog (**Tools > IP Catalog**), the Intel Quartus Prime software automatically connects the IP core to the SDM in the FPGA.

Figure 134. Intel Stratix 10 Partial Reconfiguration Controller



The SDM performs authentication, decryption, and decompression on the configuration data. The Quartus Prime software currently supports PR over the core interface using the Intel Stratix 10 PR Controller IP core, or PR over the JTAG device pins. PR over JTAG does not require the Intel Stratix 10 PR Controller IP core.

The Intel Stratix 10 PR Controller IP core interfaces with the SDM to manage the bitstream source. Instantiate this IP core in an Intel Stratix 10 design when performing partial reconfiguration using an internal PR host, Nios II, PCI Express, or Ethernet. Instantiate the IP core from the Intel Quartus Prime IP Catalog or Platform Designer.

### 8.4.6 Promoting Global Signals in a PR Region

In standard designs, the Intel Quartus Prime software automatically promotes high fan-out signals onto dedicated global networks. This global promotion happens during the Plan stage of design compilation.



In PR designs, the Compiler disables global promotion for signals originating within the logic of a PR region. Instantiate the clock control blocks only in the static region, because the clock floorplan and the clock buffers must be a part of the static region of the design. Manually instantiating a clock control block in a PR region, or assigning a signal in a PR region with the GLOBAL\_SIGNAL assignment, results in compilation error. To drive a signal originating from the PR region onto a global network:

1. Expose the signal from the PR region.
2. Drive the signal onto the global network from the static region
3. Drive the signal back into the PR region.

You can drive a maximum of 33 clocks (for Intel Arria 10 devices) or 32 clocks (for Intel Stratix 10 devices) into any PR region, but you cannot share a row clock between two PR regions. Use the Chip Planner to visualize the row clock region boundaries, and to ensure that no two PR regions share a row clock region.

When promoting global signals, the Compiler allows only certain signals to be global inside the PR regions. Use only global signals to route certain secondary signals into a PR region. The following table lists the restriction for each block:

**Table 75. Supported Signal Types for Driving Clock Networks in a PR Region**

| Block Type      | Supported Global Network Signals                       |
|-----------------|--------------------------------------------------------|
| LAB, MLAB       | Clock, ACLR, SCLR <sup>(7)</sup>                       |
| RAM, ROM (M20K) | Clock, ACLR, Write Enable (WE), Read Enable (RE), SCLR |
| DSP             | Clock, ACLR, SCLR                                      |

### 8.4.7 Partial Reconfiguration Process Sequence

The partial reconfiguration design initiates the PR operation, and delivers the configuration file to the PR control block or SDM as part of the system level design. Before partial reconfiguration, ensure that the FPGA device is in user mode, and in a functional state. The following steps describe the PR sequence:

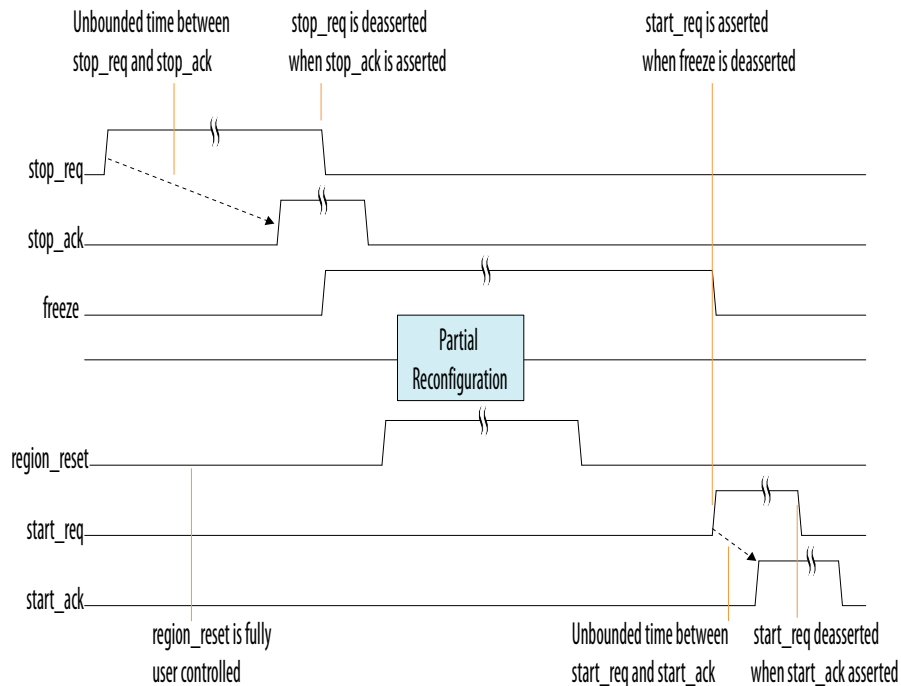
1. Send a `stop_req` signal to the PR region from the sequential PR control logic to prepare for the PR operation. This signal informs the PR regions to complete any pending transactions and stop accepting new transactions.
2. Wait for the `stop_ack` signal to indicate that the PR region is ready for partial reconfiguration.
3. Use PR control logic to freeze all necessary outputs of the PR regions. Additionally, drive the clock enable for any initialized RAMs to disabled state.
4. To initiate the PR process for the PR region, send the PR bitstream to the PR control block or SDM. When using the Partial Reconfiguration IP core, the Avalon-MM or Avalon-ST interface on the IP core handles the process. When directly instantiating the PR control block for Intel Arria 10 designs, follow the PR control block interface protocol timings to ensure that the PR process progresses correctly.
5. On successful completion of the PR operation, reset the PR region.

---

<sup>(7)</sup> Only Intel Stratix 10 designs support global SCLR.

6. Signal the PR region to start operating by asserting the `start_req` signal, and deasserting the `freeze` signal.
7. Wait for the `start_ack` signal to indicate that the PR region is ready for operation.
8. Resume operation of the FPGA with the newly reconfigured PR region.

**Figure 135. Recommended Process Sequence Timing Diagram**



### 8.4.8 Resetting the PR Region Registers

Upon partial reconfiguration of a PR region, the status of the PR region registers become indeterminate. Bring the registers in the PR region to a known state by applying a reset sequence for the PR region. This reset ensures that the system behaves to your specifications. Simply reset the control path of the PR region, if the datapath eventually flushes out within a finite number of cycles.

**Table 76. Supported PR Reset Implementation Guideline**

| PR Reset Type    | Active-High Synchronous Reset                                                                             | Active-High Asynchronous Reset | Active-Low Synchronous Reset                                                                              | Active-Low Asynchronous Reset |
|------------------|-----------------------------------------------------------------------------------------------------------|--------------------------------|-----------------------------------------------------------------------------------------------------------|-------------------------------|
| On local signal  | Yes                                                                                                       | Yes                            | Yes                                                                                                       | Yes                           |
| On global signal | <ul style="list-style-type: none"> <li>• No (Intel Arria 10)</li> <li>• Yes (Intel Stratix 10)</li> </ul> | Yes                            | <ul style="list-style-type: none"> <li>• No (Intel Arria 10)</li> <li>• Yes (Intel Stratix 10)</li> </ul> | Yes                           |

**Note:** Use active-high local reset instead of active-low, wherever applicable. This technique allows you to automatically hold the PR region in reset, by virtue of the boundary port wire LUT.



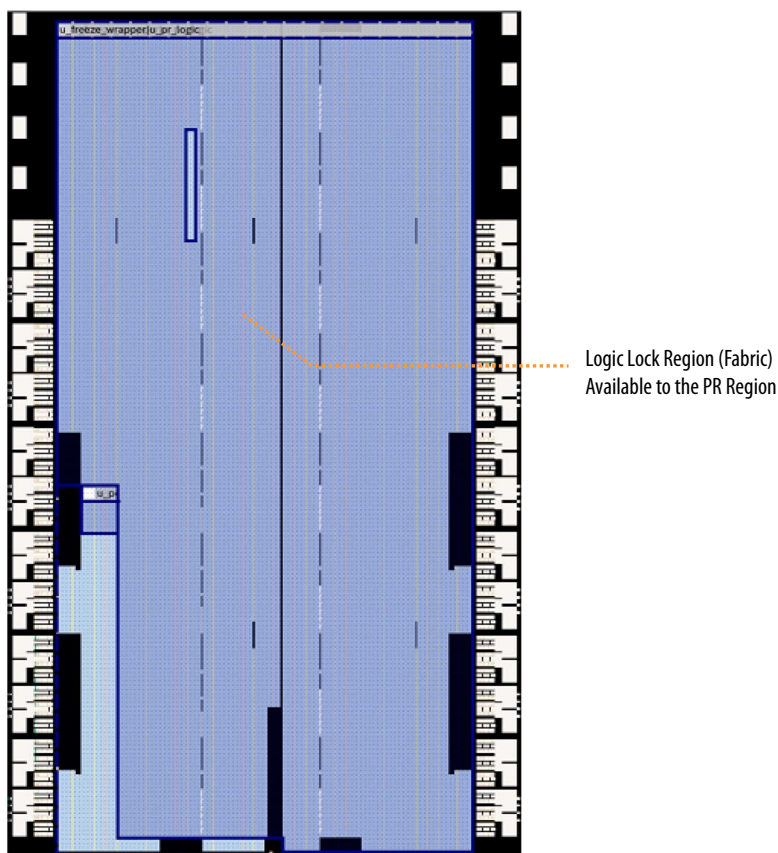
### 8.4.9 Floorplanning a Partial Reconfiguration Design

The floorplan constraints in your partial reconfiguration design physically partition the device. This partitioning ensures that the resources available to the PR region are the same for any persona that you implement.

*Note:* Complete the periphery and clock floorplan before core floorplanning.

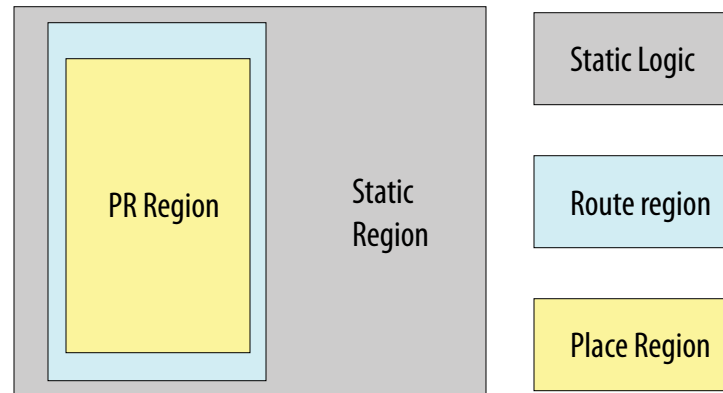
Your PR region must include only core logic, such as LABs, RAMs, ROMs, and DSPs in a PR region. Intel Stratix 10 designs can also include Hyper-Registers in the PR partition. Instantiate all periphery design elements, such as transceivers, external memory interfaces, and clock networks in the static region of the design. Logic Lock regions can cross periphery locations, such as the I/O columns and the HPS, because the constraint is core-only. The following figure shows the PR region floorplan covering the I/O columns in the middle of the device:

**Figure 136. PR Region Floorplan**



To create periphery floorplan assignments for your design, use the Interface Planner (**Tools > Interface Planner**).

Figure 137. Floorplanning your PR Design



Each PR partition in your design must have a corresponding, exclusive physical partition. Assign any Logic Lock regions to define the physical partition for your PR region. There are two region types:

- Placement regions—use these regions to constrain logic to a specific area of the device. The Fitter places the logic in the region you specify. The Fitter can also place other logic in the region unless you designate the region as **Reserved**.
- Routing regions—use these regions to constrain routing to a specific area.

The routing region must fully enclose the placement region. Additionally, the routing regions for the PR regions cannot overlap.

Create Logic Lock regions from the Project Navigator, Logic Lock Regions window, or Chip Planner. For complete information on creating Logic Lock regions, refer to *Creating Logic Lock Regions* in the Intel Quartus Prime Pro Edition Handbook.

Follow these guidelines when floorplanning your PR design:

- Define a routing region that is at least 1 unit larger than the placement region in all directions.
- Do not overlap the routing regions of multiple PR regions.
- Select the PR region row-wise for least bitstream overhead. In Intel Arria 10 devices, the short, wide regions have smaller bitstream size than tall, narrow regions. Intel Stratix 10 configuration occurs on sectors. For the least bitstream overhead, ensure that you align the PR region to sectors.
- Define sub Logic Lock regions within PR regions to improve timing closure.
- For Intel Arria 10, the height of your floorplan affects the reconfiguration time. A floorplan larger in the Y direction takes longer to reconfigure. This condition does not apply to Intel Stratix 10 devices because they configure according to sectors.
- If your design includes HPR parent and child partitions, placement region of the parent region must fully enclose the routing and placement region of its child region. Also, the parent wire LUTs must be in an area, outside the child PR region. This requirement is because the child PR region is exclusive to all other logic, which includes the parent and the static region.



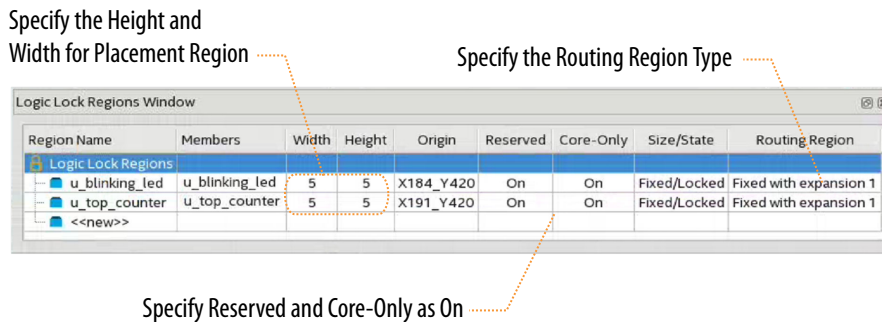


To create a Logic Lock region for your PR partition:

1. Right-click the design instance in the **Project Navigator** and click **Logic Lock Region > Create New Logic Lock Region**. The region appears on the Logic Lock Regions Window.
2. In the Logic Lock Regions window, specify the placement region coordinates in the **Origin** column. The origin corresponds to the lower-left corner of the region. For example, to set a placement region with lower-left corner co-ordinates of (184 420), specify the **Origin** as X184\_Y420. The Compiler automatically calculates the upper-right corner coordinates for the placement region, based on the height and width you specify.
3. Enable the **Reserved** and **Core-Only** options for the region.
4. Double-click the **Routing Region** option. The **Logic Lock Routing Region Settings** dialog box appears.
5. Specify the **Routing type**. The Logic Lock region supports the following routing types:
  - **Whole chip**—allocates the entire chip for the routing shape.
  - **Fixed with expansion**—allocates an expansion length of 1 for the routing shape.
  - **Custom**—allows you to manually add a custom routing shape and specify the **Height, Width, and Origin**.

*Note:* The routing shape must be larger than the placement shape.
6. Click **OK**.

Figure 138. Logic Lock Regions Window



The following assignments in the .qsf file correspond to creating a core-only, reserved Logic Lock region with placement and routing regions:

```
set_instance_assignment -name PARTITION supr_partition -to u_top_counter
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to \
u_top_counter
set_instance_assignment -name PLACE_REGION "X191 Y420 X195 Y424" -to \
u_top_counter
set_instance_assignment -name RESERVE_PLACE_REGION ON -to u_top_counter
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to u_top_counter
set_instance_assignment -name ROUTE_REGION "X190 Y419 X196 Y425" -to \
u_top_counter

set_instance_assignment -name PARTITION pr_partition -to u_blinking_led
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to \
u_blinking_led
```

### Related Links

- [Analyzing and Optimizing the Design Floorplan](#)  
For complete information on how to create Logic Lock regions.
- [Interface Planner](#)  
For complete information on interface planning.

#### 8.4.9.1 Applying Floorplan Constraints Incrementally

PR implementation requires additional constraints that identify the reconfigurable partitions of the design and device. These constraints significantly impact the Compiler's timing closure ability. You can avoid and more easily correct timing closure issues by incrementally implementing each constraint, running the Compiler, then verifying timing closure.

*Note:* PR designs require a more constrained floorplan, compared to a flat design. The overall density and performance of a PR design may be lower than an equivalent flat design.

The following steps describe incrementally developing the requirements for your PR design:

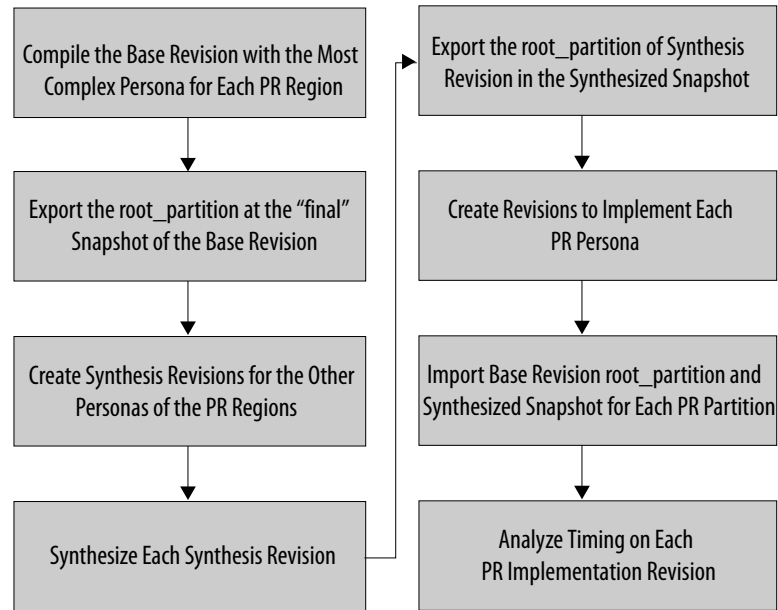
1. Implement the base revision using the most complex persona for each PR partition. This initial implementation must include the complete design with all periphery constraints, and top-level `.sdc` timing constraints. Do not include any Logic Lock region constraints for the PR regions with this implementation.
2. Create partitions by setting the region **Type** option to **Default** in the Design Partitions Window, for all the PR partitions.
3. Register the boundaries of each partition to ensure adequate timing margin.
4. Verify successful timing closure using the Timing Analyzer.
5. Ensure that all the desired signals are driven on global networks. Disable the **Auto Global Clock** option in the Fitter (**Assignments > Settings > Compiler Settings > Advanced Settings (Fitter)**), to avoid promoting non-global signals.
6. Create Logic Lock core-only placement regions for each of the partitions.
7. Recompile the base revision with the Logic Lock constraints, and then verify timing closure.
8. Enable the **Reserved** option for each Logic Lock region to ensure the exclusive placement of the PR partitions within the placement regions. Enabling the **Reserved** option avoids placing the static region logic in the placement region of the PR partition.
9. Recompile the base revision with the **Reserved** constraint, and then verify timing closure.
10. In the Design Partitions Window, specify each of the PR partitions as the **Reconfigurable Type**. This assignment ensures that the Compiler adds wire LUTs for each interface of the PR partition, and performs additional compilation checks for partial reconfiguration.
11. Recompile the base revision with the **Reconfigurable** constraint, and then verify timing closure. You can now export the top-level partition for reuse in the PR implementation compilation of the different personas.



### 8.4.10 Creating Revisions for Personas

To compile a partial reconfiguration project, create a base revision for the design. Also, create synthesis and PR implementation revisions for each of the personas.

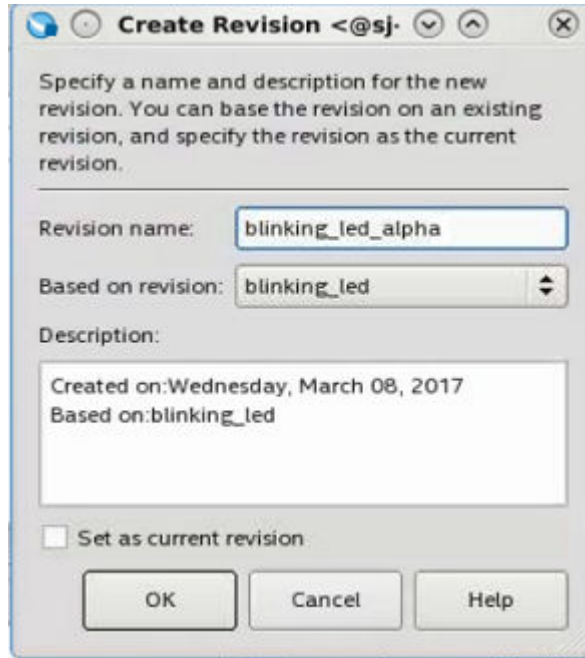
**Figure 139. Partial Reconfiguration Compilation Flow for Intel Arria 10 and Intel Stratix 10 Devices**



To create the PR implementation revisions:

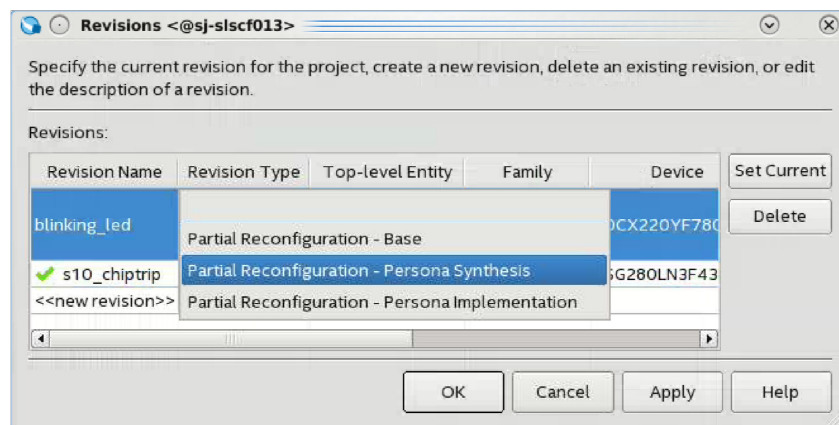
1. Click **Project > Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. Specify a unique **Revision name**.
4. Select an existing revision for the **Based on revision** option.
5. Enable **Set as current revision** to specify the persona as your current revision, and click **OK**.

Figure 140. Create Revisions Dialog Box

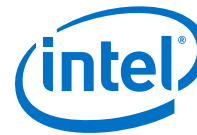


6. To set the revision type, click **Assignments > Settings > General**.
7. Select the persona for setting the revision type from the **Recently selected top-level entities** list.
8. Select the revision type:
  - **Partial Reconfiguration - Base**
  - **Partial Reconfiguration - Persona Synthesis**
  - **Partial Reconfiguration - Persona Implementation**

Figure 141. Specify Revision Type



9. Click **Apply** and **OK**.



The following assignments in the respective revision's .qsf file correspond to specifying the revision type from the **Settings** dialog box:

```
set_global_assignment -name REVISION_TYPE PR_BASE
set_global_assignment -name REVISION_TYPE PR_SYN
set_global_assignment -name REVISION_TYPE PR_IMPL
```

### 8.4.11 Compiling the Partial Reconfiguration Design

The number of compilations a PR design requires depends on the number of PR personas. Use the base revision compilation, and each PR implementation compilation for timing analysis.

Typically, compile a partial reconfiguration design in two phases:

1. From the Compilation Dashboard, run the **Place** and **Route** compilation stages on the static partitions, along with a set of default personas for each PR partition.
2. Compile the alternate personas, while preserving the static partition's placing and routing blocks.

When reusing or preserving a design block, always specify the precise compilation snapshot to reuse. For example, when compiling the alternate personas of a PR design, specify the snapshot for that compilation as the "final" snapshot of the static region. Otherwise, the Compiler cannot preserve the routing information.

#### Related Links

##### [Design Compilation](#)

For more information on how to analyze, synthesize, place, and route your design.

#### 8.4.11.1 Generating the Partial Reconfiguration Flow Script

The Intel Quartus Prime Pro Edition software provides a flow template for compiling a partial reconfiguration design for Intel Arria 10 and Intel Stratix 10 devices.

To create and run a PR flow script in your Intel Quartus Prime project directory, follow these steps:

1. Type one of the following commands from the Intel Quartus Prime shell:

```
quartus_sh --write_flow_template -flow s10_partial_reconfig
```

```
quartus_sh --write_flow_template -flow s10_hier_partial_reconfig
```

```
quartus_sh --write_flow_template -flow a10_partial_reconfig
```

```
quartus_sh --write_flow_template -flow a10_hier_partial_reconfig
```

2. To run the script type one of the following commands:

```
quartus_sh -t s10_partial_reconfig/flow.tcl
```

```
quartus_sh -t s10_hier_partial_reconfig/flow.tcl
```

```
quartus_sh -t a10_partial_reconfig/flow.tcl
```

```
quartus_sh -t a10_hier_partial_reconfig/flow.tcl
```

Use the following options when running this script:

**Table 77. Partial Reconfiguration Flow Script Options**

| Option                      | Description                                                                                                                                                                                |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -all                        | Default option that compiles the base revision and all the PR implementation revisions.                                                                                                    |
| --impl[=<name>]             | Compiles a specified PR implementation. Specify the revision name of the implementation to compile.                                                                                        |
| -all_impl                   | Compiles all PR implementations. Skips the base revision compilation.                                                                                                                      |
| -base                       | Compiles the base revision. Skips all the PR implementations compilation.                                                                                                                  |
| -check                      | Checks the script configuration and exits without performing any compilation.                                                                                                              |
| -setup_script[=<file_name>] | Allows you to customize the script settings with your partial reconfiguration project details. The settings you define in this file override the variable settings in the script template. |

#### 8.4.11.1.1 Configuring the Partial Reconfiguration Flow Script

To configure the PR flow script for your design:

1. Rename the generated PR flow script:

**Table 78. PR Script Renaming**

| Rename This File                       | To This File                   |
|----------------------------------------|--------------------------------|
| a10_partial_reconfig/setup.tcl.example | a10_partial_reconfig/setup.tcl |
| s10_partial_reconfig/setup.tcl.example | s10_partial_reconfig/setup.tcl |

2. Edit the `setup.tcl` file with configuration that overrides the variable settings in the `a10_partial_reconfig/flow.tcl` or `s10_partial_reconfig/flow.tcl` file. To define the name of your Intel Quartus Prime partial reconfiguration project, modify the following line:

```
define_project <project_name>
```

*Note:* All revisions must be present in the corresponding `.qpf` file.

3. To define the base revision name, modify the following line:

```
define_base_revision <base_revision_name>
```

This revision represents the static region of the design.

4. To define each of the partial reconfiguration implementation revisions, along with the PR partition names and the synthesis revision that implements the revisions, modify the following line:

```
define_pr_impl_partition -impl_rev_name <implementation_revision_name> \  
-partition_name <pr_partition_name\  
-source_rev_name <synthesis_revision_name>  
...  
...
```

*Note:* Alternatively, use the `setup_script` option while running the `flow.tcl` script to specify the `setup.tcl` configuration file location.



#### 8.4.11.1.2 Running the Partial Reconfiguration Flow Script

To run the partial reconfiguration flow script with your setup file:

1. Click **Tools** > **Tcl Scripts**. The **Tcl Scripts** dialog box appears.
2. Click **Add to Project**, browse and select `a10_partial_reconfig/flow.tcl` or `s10_partial_reconfig/flow.tcl`
3. Select `a10_partial_reconfig/flow.tcl` or `s10_partial_reconfig/flow.tcl` in the **Libraries** pane, and then click **Run**.

Alternatively, to run the script from the Intel Quartus Prime command shell, type one of the following commands:

```
quartus_sh -t a10_partial_reconfig/flow.tcl -setup_script setup.tcl
```

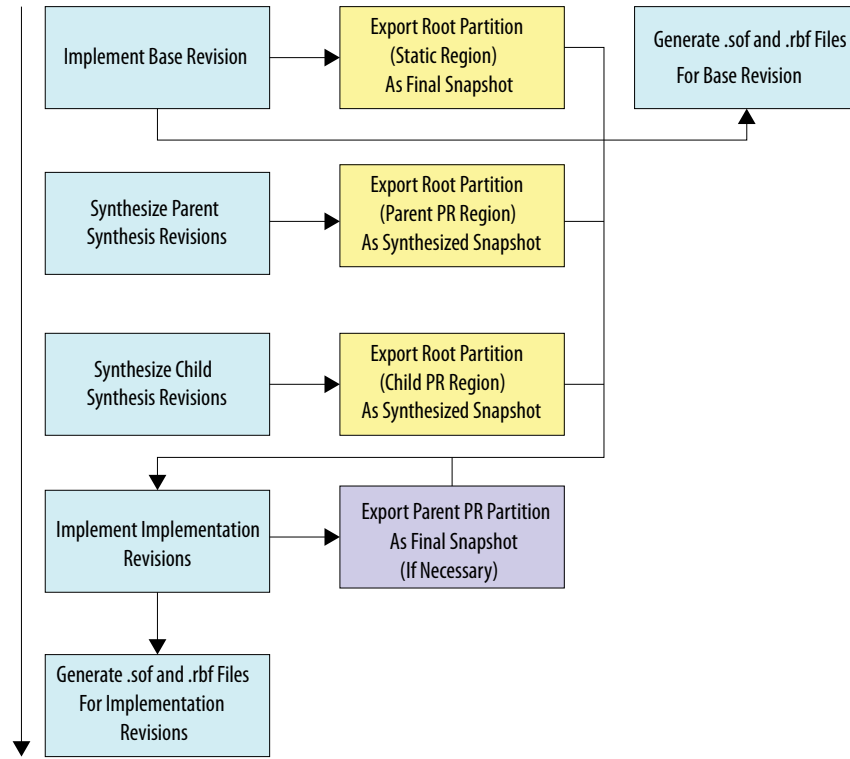
```
quartus_sh -t s10_partial_reconfig/flow.tcl -setup_script setup.tcl
```

#### 8.4.11.2 Hierarchical Partial Reconfiguration Compilation Flow

Intel Arria 10 and Intel Stratix 10 devices support hierarchical partial reconfiguration (HPR), which allows multiple parent and child design partitions, or multiple levels of partitions in the same design. Similar to compiling a standard PR design, you must create a base revision for your design. Then, you create dedicated synthesis revisions for each parent and child partition in your design. Specify the design file of the parent and child partition as the top-level entity for the corresponding synthesis revision. Also, ensure that the synthesis revisions for the parent PR partitions include the partition assignments for its child PR regions.

When compiling the implementation revision for an HPR design, you must fully floorplan the child partition, in the same manner as the PR region of a base revision.

**Figure 142. Hierarchical Partial Reconfiguration Compilation Flow**



#### 8.4.11.2.1 Configuring the Hierarchical Partial Reconfiguration Flow Script

To configure the HPR flow script for your design:

1. Rename the generated script files:

**Table 79. PR Script Renaming**

| Rename This File                            | To This File                        |
|---------------------------------------------|-------------------------------------|
| a10_hier_partial_reconfig/setup.tcl.example | a10_hier_partial_reconfig/setup.tcl |
| s10_hier_partial_reconfig/setup.tcl.example | s10_hier_partial_reconfig/setup.tcl |

2. Edit the `setup.tcl` file with a configuration that overrides the variable settings in the `a10_hier_partial_reconfig/flow.tcl` or `s10_hier_partial_reconfig/flow.tcl` file. To define the name of your Intel Quartus Prime hierarchical partial reconfiguration project, modify the following line:

```
define_project <project_name>
```

*Note:* All revisions must be present in the corresponding `.qpf` file.

3. To define the base revision name, modify the following line:

```
define_base_revision <base_revision_name>
```





This revision represents the static region of the design.

- To define each parent and child partition in each revision, along with the partition names, the implementation revision name, source revision name, source revision partition name, and the source snapshot, modify the following lines:

```
define_pr_impl_partition -impl_rev_name <implementation revision name> \  
-partition_name <partition_name> \  
-source_rev_name <source_revision_name> \  
-source_partition <source_partition_name> \  
-source_snapshot <source_snapshot>
```

*Note:* Alternatively, use the `setup_script` option while running the `flow.tcl` script to specify the `setup.tcl` configuration file location.

**Table 80. Implementation Revision Definition Arguments**

| Argument          | Description                                                                                                                                                                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -impl_rev_name    | Defines the implementation revision name.                                                                                                                                                                                                                                                           |
| -partition_name   | Defines the partition name.                                                                                                                                                                                                                                                                         |
| -source_rev_name  | Defines the name of the source revision. This revision can be a synthesis or implementation revision, from which this partition imports the exported synthesis/final snapshot, for implementation.                                                                                                  |
| -source_partition | Defines the partition in the source revision, which the Compiler exports, later in the flow. This partition can either be a root partition for synthesis source revisions, or a parent PR partition for implementation source revisions.                                                            |
| -source_snapshot  | Defines the snapshot of the source partition that the Compiler exports, later in the flow. Usually, you define this argument as the final snapshot for parent PR partitions exported from implementation revisions, and synthesized snapshot for root partitions exported from synthesis revisions. |

*Note:* Alternatively, use the `setup_script` option while running the `flow.tcl` script to specify the `setup.tcl` configuration file location.

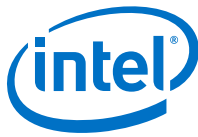
#### Related Links

- [AN 805: Hierarchical Partial Reconfiguration of a Design on Intel Arria 10 SoC Development Board](#)
- [AN 816: Hierarchical Partial Reconfiguration Tutorial for Intel Stratix 10 GX FPGA Development Board](#)

#### 8.4.11.2.2 Running the Hierarchical Partial Reconfiguration Flow Script

To run the hierarchical partial reconfiguration flow script with your setup file:

- Click **Tools** ► **Tcl Scripts**. The **Tcl Scripts** dialog box appears.
- Click **Add to Project**, browse and select the `a10_hier_partial_reconfig/flow.tcl` or `s10_hier_partial_reconfig/flow.tcl` file.
- Select the `a10_hier_partial_reconfig/flow.tcl` or `s10_hier_partial_reconfig/flow.tcl` in the Libraries pane, and click **Run**.



Alternatively, to run the script from the Intel Quartus Prime command shell, type one of the following commands:

```
quartus_sh -t a10_hier_partial_reconfig/flow.tcl -setup_script /  
a10_hier_partial_reconfig/setup.tcl
```

```
quartus_sh -t s10_hier_partial_reconfig/flow.tcl -setup_script /  
s10_hier_partial_reconfig/setup.tcl
```

**Table 81. Hierarchical Partial Reconfiguration Flow Script Options**

| Option                      | Description                                                                                                                                                                                                                                                |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -all                        | Default option that compiles the base revision and all the PR implementation revisions.                                                                                                                                                                    |
| -all_syn                    | Compiles all the HPR synthesis revisions. Skips the base revision compilation.                                                                                                                                                                             |
| -impl[=<name>]              | Compiles a specified HPR implementation revision. Specify the revision name of the implementation to compile.                                                                                                                                              |
| -all_impl                   | Compiles all HPR implementations. Skips the base revision compilation.                                                                                                                                                                                     |
| -base                       | Compiles the base revision. Skips all the HPR implementations compilation.                                                                                                                                                                                 |
| -check                      | Checks the script configuration and exits without performing any compilation.                                                                                                                                                                              |
| -setup_script[=<file_name>] | Allows you to customize the script settings with your partial reconfiguration project details. The settings you define in this file override the variable settings in a10_hier_partial_reconfig/setup.tcl or s10_hier_partial_reconfig/setup.tcl template. |

## 8.4.12 Timing Analysis with Partial Reconfiguration

The interface between partial and static partitions remains the same for each PR implementation revision. Perform timing analysis on each PR implementation revision to ensure that there are no timing violations.

To meet various timing requirements for multiple PR personas, create separate .sdc files for each persona. If you need timing constraints for the synthesis persona, include the constraints in the synthesis revision. The target name must match the hierarchy of the persona at the top-level.

**Note:** Logic Lock regions impose placement constraints that affect the performance and resource utilization of your PR design. Ensure that the design has additional timing allowance and available device resources. Selecting the largest and most timing-critical persona as your base persona optimizes the timing closure.

### Related Links

[The Intel Quartus Prime Timing Analyzer](#)

### 8.4.12.1 Running Timing Analysis on a Design with PR Partitions

To ensure timing closure of designs with multiple PR regions, you create aggregate revisions for all possible PR region combinations.



To create an aggregate revision and perform timing analysis on the aggregate revision:

1. To open the **Revisions** dialog box, click **Project > Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. Specify the **Revision name** and select the base revision for **Based on Revision**.
4. Ensure that you include all the `.sdc` and `.ip` files for the static and PR region.

*Note:* To detect the clocks, the `.sdc` file for the PR Controller IP must follow any `.sdc` that creates the clocks that the IP core uses. You facilitate this order by ensuring the `.ip` file for the PR Controller IP core comes after any `.ip` files or `.sdc` files that you use to create these clocks in the `.qsf` file for your Intel Quartus Prime project revision. For more information, refer to the *Partial Reconfiguration Solutions IP User Guide*.

5. To export the post-fit database from the base compile (static partition), type the following command in the Intel Quartus Prime shell:

```
quartus_cdb <base_revision> --export_block "root_partition" --snapshot \
  final --file "<base revision name>.qdb" --exclude_pr_subblocks
```

*Note:* The static partition post-fit database is already available in the base revision. You can use this `<base revision name>.qdb` file from the base revision project folder, instead of regenerating the `.qdb` file using the above command.

6. To export the post-fit database from the multiple personas (PR implementation revisions), type the following commands in the Intel Quartus Prime shell:

```
quartus_cdb <PR1 Fit revision> --export_block <PR1 Partition name> \
  --snapshot final --file "pr1.qdb"

quartus_cdb <PR2 Fit revision> --export_block <PR2 Partition name> \
  --snapshot final --file "pr2.qdb"
```

7. To import the post-fit databases of the static and PR region as aggregate revision, type the following commands in the Intel Quartus Prime shell:

```
quartus_cdb <aggr_rev> --import_block "root_partition" --file \
  "<base revision name>.qdb"

quartus_cdb <aggr_rev> --import_block <PR1 partition name> --file "pr1.qdb"

quartus_cdb <aggr_rev> --import_block <PR2 Partition name> --file "pr2.qdb"
```

8. To integrate post-fit database of all the partitions, type the following command in the Intel Quartus Prime shell:

```
quartus_fit <proj name> -c <aggr_rev>
```

*Note:* The Fitter verifies the legality of the post-fit database, and combines the netlist for timing analysis. The Fitter does not reroute the design.

9. To perform timing analysis on the aggregate revision, type the following command in the Intel Quartus Prime shell:

```
quartus_sta <proj name> -c <aggr_rev>
```

10. Run timing analysis on aggregate revision for all possible PR persona combination. If a specific persona fails timing closure, recompile the persona and perform timing analysis again.

## Related Links

[Partial Reconfiguration Solutions IP User Guide](#)

### 8.4.13 External Host Configuration (Intel Arria 10 Designs Only)

The current version of the Intel Quartus Prime Pro Edition software supports external host PR configuration only for Intel Arria 10 devices. During user mode, the external host initiates partial reconfiguration, and monitors the PR status using the external PR dedicated pins. In this mode, the external host must respond appropriately to the handshake signals for successful partial reconfiguration. The external host writes the partial bitstream data from external memory into the Intel Arria 10 device. Coordinate system-level partial reconfiguration by ensuring that you prepare the correct PR region for partial reconfiguration. After reconfiguration, return the PR region into operating state.

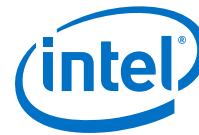
To use an external host for your design:

1. Click **Assignments > Device > Device & Pin Options**.
2. Select the **Enable PR Pins** option in the **Device & Pin Options** dialog box. This option automatically creates the special partial reconfiguration pins, and defines the pins in the device pin-out. This option also automatically connects the pins to PR control block internal path.  
*Note:* If you do not select this option, you must use an internal or HPS host. You do not need to define pins in your design top-level entity.
3. Connect these top-level pins to the specific ports in the PR control block.

The following table lists the automatically constrained PR pins when you select **Enable PR Pins** option, and the specific PR control block port to connect these pins to:

**Table 82. Partial Reconfiguration Dedicated Pins**

| Pin Name   | Type   | PR Control Block Port Name | Description                                                                                           |
|------------|--------|----------------------------|-------------------------------------------------------------------------------------------------------|
| PR_REQUEST | Input  | prrequest                  | Logic high on this pin indicates that the PR host is requesting partial reconfiguration.              |
| PR_READY   | Output | ready                      | Logic high on this pin indicates that the PR control block is ready to begin partial reconfiguration. |
| PR_DONE    | Output | done                       | Logic high on this pin indicates that the partial reconfiguration is complete.                        |
| PR_ERROR   | Output | error                      | Logic high on this pin indicates an error in the device during partial reconfiguration.               |
| DATA[31:0] | Input  | data                       | These pins provide connectivity for PR_DATA to transfer the PR bitstream to the PR controller.        |
| DCLK       | Input  | clk                        | Receives synchronous PR_DATA.                                                                         |



- Note:**
1. PR\_DATA can be 8, 16, or 32-bits in width.
  2. Ensure that you connect the `corectl` port of the PR control block to 0.

### Example 71. Verilog RTL for External Host PR

```

module top(
    // PR control block signals
    input logic pr_clk,
    input logic pr_request,
    input logic [31:0] pr_data,
    output logic pr_error,
    output logic pr_ready,
    output logic pr_done,

    // User signals
    input logic i1_main,
    input logic i2_main,
    output logic o1
);

// Instantiate the PR control block
twentynm_prblock m_prblock
(
    .clk(pr_clk),
    .corectl(1'b0),
    .prrequest(pr_request),
    .data(pr_data),
    .error(pr_error),
    .ready(pr_ready),
    .done(pr_done)
);

// PR Interface partition
pr_v1 pr_inst(
    .i1(i1_main),
    .i2(i2_main),
    .o1(o1)
);
endmodule

```

### Example 72. VHDL RTL for External Host PR

```

library ieee;
use ieee.std_logic_1164.all;

entity top is
port(
    -- PR control block signals
    pr_clk: in std_logic;
    pr_request: in std_logic;
    pr_data: in std_logic_vector(31 downto 0);

    pr_error: out std_logic;
    pr_ready: out std_logic;
    pr_done: out std_logic;

    -- User signals
    i1_main: in std_logic;
    i2_main: in std_logic;
    o1: out std_logic
);
end top;

architecture behav of top is
component twentynm_prblock is

```

```
port(
    clk: in std_logic;
    corectl: in std_logic;
    prrequest: in std_logic;
    data: in std_logic_vector(31 downto 0);
    error: out std_logic;
    ready: out std_logic;
    done: out std_logic
);
end component;

component pr_v1 is
port(
    i1: in std_logic;
    i2: in std_logic;
    o1: out std_logic
);
end component;

signal pr_gnd : std_logic;

begin

pr_gnd <= '0';

-- Instantiate the PR control block
m_prblock: twentynm_prblock port map
(
    pr_clk,
    pr_gnd,
    pr_request,
    pr_data,
    pr_error,
    pr_ready,
    pr_done
);

-- PR Interface partition
pr_inst : pr_v1 port map
(
    i1_main,
    i2_main,
    o1
);

end behavior;
```

### 8.4.14 Programming File Generation

The Intel Quartus Prime Pro Edition Assembler generates the PR bitstreams for your design personas. For Intel Arria 10 designs, you send the bitstreams to the PR control block. For Intel Stratix 10 designs, you send the bitstreams to the SDM. You must compile the PR project, including the base revision, and at least one reconfigurable revision, before generating the PR bitstreams.

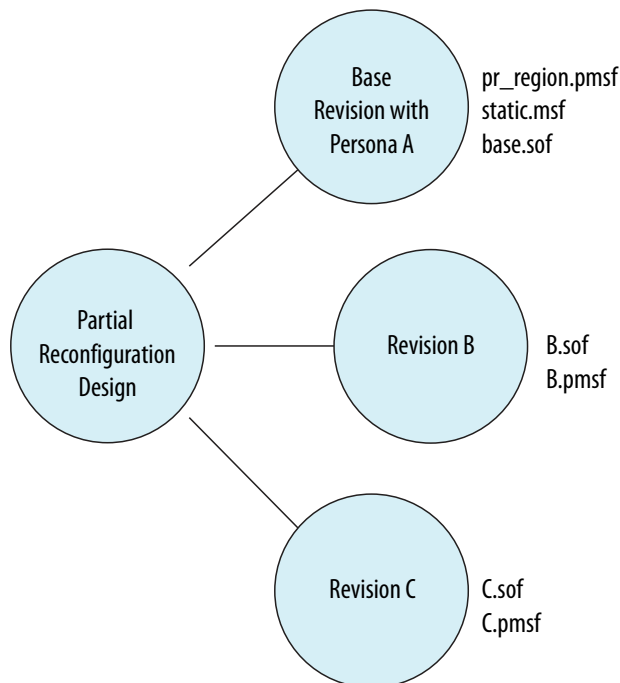
**Note:** For Intel Stratix 10 designs, the Assembler generates a configuration `.rbf` automatically at the end of compilation. For Intel Arria 10 designs, you must click **File > Convert Programming Files** to convert the Partial-Masked SRAM Object Files (`.pmsf`) that the Assembler generates to an `.rbf` file.



### Example 73. Programming File Generation for a Partial Reconfiguration Design

This example design contains a PR region and the following revisions:

- Base revision with persona A
- PR revision with persona B
- PR revision with persona C



**Table 83. Programming Files for PR Designs**

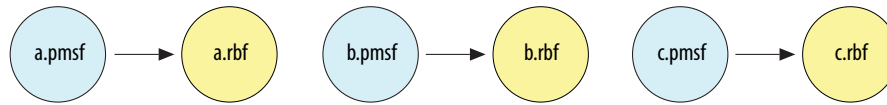
| Programming File          | Description                                                                                                                                                                                        |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <rev>.<pr_region>.pmsf    | Contains the partial-mask bits for the PR region. The .pmsf file contains all the information for creating PR bitstreams.<br><i>Note:</i> The default file name corresponds to the partition name. |
| <rev>.<static_region>.msf | Contains the mask bits for the static region.                                                                                                                                                      |
| <rev>.sof                 | Contains configuration information for the entire device.                                                                                                                                          |

#### 8.4.14.1 Generating PR Bitstreams

After creating the .pmsf files, process the PR bitstreams to generate the Raw Binary File (.rbf) files for reconfiguration. Convert the .pmsf file for every PR region in your Intel Arria 10 design to .rbf file format. Using the .rbf format stores the bitstream in an external flash memory.

*Note:* If you use the partial reconfiguration flow script, the script automatically performs the following steps according to the options you set.

Figure 143. Generating PR Bitstreams



To generate the .rbf file:

1. Click **File** ► **Convert Programming Files**. The **Convert Programming Files** dialog box appears.
2. Specify the output file name and **Programming file type** as **Raw Binary File for Partial Reconfiguration (.rbf)**.
3. To add the input .pmsf file to convert, click **Add File**.
4. Select the newly added .pmsf file, and click **Properties**.
5. For Intel Arria 10 designs, enable or disable any of the following options:
  - **Compression**—enables compression on PR bitstream.
  - **Enhanced compression**—enables enhanced compression on PR bitstream.
  - **Generate encrypted bitstream**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (.ekp).

*Note:* Enabling the **Generate encrypted bitstream** option automatically disables **Compression** and **Enhanced compression**. Conversely, enabling **Compression** or **Enhanced compression** automatically disables **Generate encrypted bitstream**. You cannot use compression and encryption at the same time.

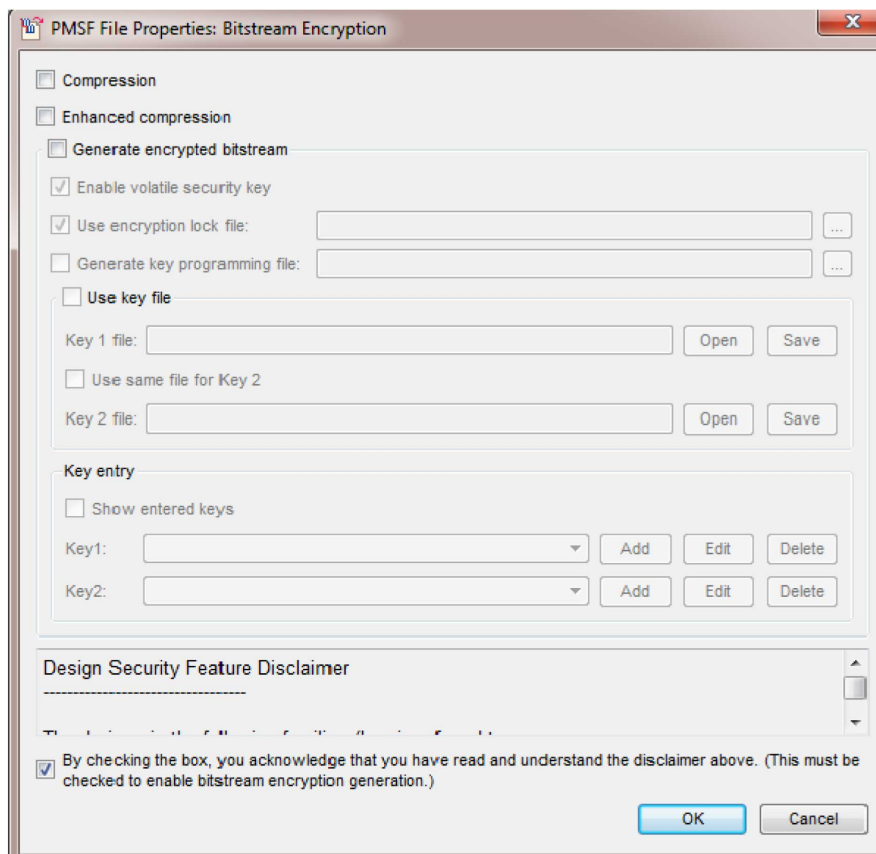
If you enable the **Generate encrypted bitstream** option, specify the following options:

- **Enable volatile security key**
- **Use encryption lock file**
- **Generate key programming file**

- Note:*
- Enabling the **Use encryption lock file** option requires that you import the encryption lock (.q1k) file generated from the base image.
  - If you configure the device using JTAG, the Programmer does not support base encryption.



Figure 144. PMSF File Properties Bitstream Encryption



Alternatively, to convert your .pmsf file to .rbf file from Intel Quartus Prime shell, type the following command:

```
quartus_cpf -c <pr_pmsf_file> <pr_rbf_file>
```

#### 8.4.14.2 Generating a Merged .pmsf File from Multiple .pmsf Files

Use a single merged .rbf file to reconfigure two PR regions simultaneously. To merge two or more .pmsf files:

1. Open the **Convert Programming Files** dialog box.
2. Specify the output file name and programming file type as **Merged Partial-Mask SRAM Object File (.pmsf)**.
3. In the **Input files to convert** dialog box, select **PMSF Data**.
4. To add input files, click **Add File**. You must specify two or more files for merging.
5. To generate the merged file, click **Generate**.

Alternatively, to merge two or more .pmsf files from the Intel Quartus Prime shell, type the following command:

```
quartus_cpf --merge_pmsf=<number of merged files> <pmsf_input_file_1> /  
<pmsf_input_file_2> <pmsf_input_file_etc> <pmsf_output_file>
```

For example, to merge two .pmsf files, type the following command:

```
quartus_cpf --merge_pmsf=<2> <pmsf_input_file_1> <pmsf_input_file_2> /
<pmsf_output_file>
```

### 8.4.14.3 CD Ratio for Bitstream Encryption and Compression (Intel Arria 10 Only)

When instantiating the Partial Reconfiguration Controller IP core in your design, you cannot use both data compression and encryption simultaneously. Enhanced decompression uses the same Clock-to-Data (CD) ratio as plain bitstreams (that is, with both encryption and compression off).

**Note:** Intel Quartus Prime software does not support both bitstream encryption and compression for Intel Stratix 10 designs.

The following table lists the valid combinations of bitstream encryption and compression. when enhance compression is enabled, always refer to x16 data width. If you use compression and enhanced compression together, the CD ratio follows the compression bitstream - 4. If you use plain and enhanced compression together, the CD ratio follows the plain bitstream - 1.

**Table 84. Valid Combinations and CD Ratio for Bitstream Encryption and Compression**

| Configuration Data Width | AES Encryption | Basic Compression | CD Ratio |
|--------------------------|----------------|-------------------|----------|
| x8                       | Off            | Off               | 1        |
|                          | Off            | On                | 2        |
|                          | On             | Off               | 1        |
| x16                      | Off            | Off               | 1        |
|                          | Off            | On                | 4        |
|                          | On             | Off               | 2        |
| x32                      | Off            | Off               | 1        |
|                          | Off            | On                | 8        |
|                          | On             | Off               | 4        |

Use the exact CD ratio that the *Valid combinations and CD Ratio for Bitstream Encryption and Compression* table specifies for different bitstream types. The CD ratio for plain .rbf must be 1. The CD ratio for compressed .rbf must be 2, 4 or 8, depending on the width. Do not specify the CD ratio as the necessary minimum to support different bitstream types.

#### 8.4.14.3.1 Generating an Encrypted PR Bitstream

To partially reconfigure your Intel Arria 10 device with encrypted bitstream:

**Note:** Intel Quartus Prime software does not support bitstream encryption and compression for Intel Stratix 10 designs.



1. Create a 256-bit key file (.key).
2. To generate the key programming file (.ekp) from the Intel Quartus Prime shell, type the following command:

```
quartus_cpf --key <keyfile>:<keyid> /  
  <base_sof_file> <output_ekp_file>
```

For example:

```
quartus_cpf --key my_key.key:key1 base.sof key.ekp
```

3. To generate the encrypted PR bitstream (.rbf), run the following command:

```
quartus_cpf -c <pr_pmsf_file> <pr_rbf_file>  
qcrypt -e --keyfile=<keyfile> --keyname=<keyid> -lockto=/  
  <clk_file> --keystore=<battery|OTP> /  
  <pr_rbf_file> <pr_encrypted_rbf_file>
```

- lockto—specifies the encryption lock.
- keystore—specifies the volatile key (battery) or the non-volatile key (OTP).

For example:

```
quartus_cpf -c top_v1.pr_region.pmsf top_v1.pr_region.rbf /  
qcrypt -e --keyfile=my_key.key --keyname=key1 --keystore=battery /  
top_v1.pr_region.rbf top_v1_encrypted.rbf
```

4. To program the key file as volatile key (default) into the device, type the following command:

```
quartus_pgm -m jtag -o P;<output_ekp_file>
```

For example:

```
quartus_pgm -m jtag -o P;key.ekp
```

5. To program the base image into the device, type the following command:

```
quartus_pgm -m jtag -o P;<base_sof_file>
```

For example:

```
quartus_pgm -m jtag -o P;base.sof
```

6. To partially reconfigure the device with the encrypted bitstream, type the following command:

```
quartus_pgm -m jtag --pr <output_encrypted_rbf_file>
```

For example:

```
quartus_pgm -m jtag --pr top_v1_encrypted.rbf
```

For more information about the design security features in Intel Arria 10 devices, refer to *Using the Design Security Features in Intel FPGAs*.

## Related Links

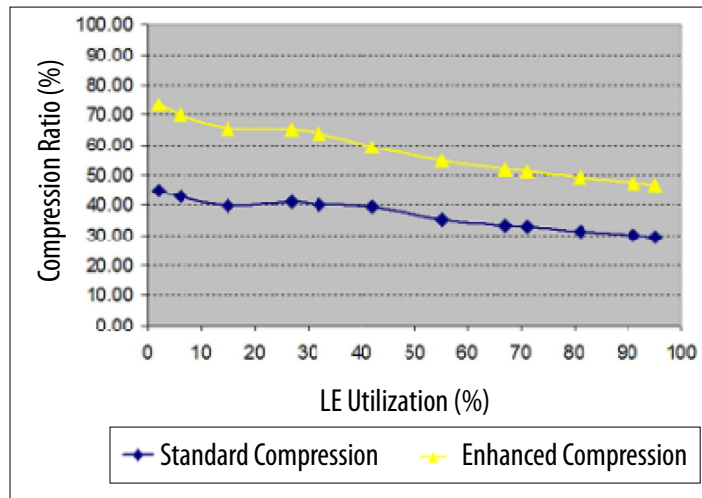
[Using the Design Security Features in Intel FPGAs](#)

### 8.4.14.3.2 Data Compression Comparison (Intel Arria 10 Designs Only)

Standard compression results in a 30-45% decrease in .rbf size. Use of the enhanced data compression algorithm results in 55-75% decrease in .rbf size. The algorithm increases the compression at the expense of additional core area required to implement the compression algorithm.

The following figure shows the compression ratio comparison across PR designs with varying degrees of Logic Element (LE):

**Figure 145. Compression Ratio Comparison between Standard Compression and Enhanced Compression**



**Note:** Intel Quartus Prime software does not support both bitstream encryption and compression for Intel Stratix 10 designs.

### 8.4.14.4 Generating Raw Binary Programming Files

The raw binary programming file (.rbf) file contains the device configuration data in little-endian raw binary format. For Intel Stratix 10 designs, the Assembler automatically generates the programming .rbf. For Intel Arria 10 designs, you generate the .rbf by converting the .pmsf file for every PR region in your design to .rbf file format.

The following examples show transmitting the .rbf byte sequence 02 1B EE 01, in x8, x16, and x32 modes respectively:

**Table 85. Writing to the PR control block or SDM in x16 mode**

In x16 mode, the first byte in the file is the least significant byte of the configuration word, and the second byte is the most significant byte of the configuration word.

|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| WORD0 = 1B02    |                 | WORD1 = 01EE    |                 |
| LSB: BYTE0 = 02 | MSB: BYTE1 = 1B | LSB: BYTE2 = EE | MSB: BYTE3 = 01 |
| D[7..0]         | D[15..8]        | D[7..0]         | D[15..8]        |
| 0000 0010       | 0001 1011       | 1110 1110       | 0000 0001       |



**Table 86. Writing to the PR control block or SDM in x32 mode**

In x32 mode, the first byte in the file is the least significant byte of the configuration double word, and the fourth byte is the most significant byte.

|                        |            |            |                 |
|------------------------|------------|------------|-----------------|
| Double Word = 01EE1B02 |            |            |                 |
| LSB: BYTE0 = 02        | BYTE1 = 1B | BYTE2 = EE | MSB: BYTE3 = 01 |
| D[7..0]                | D[15..8]   | D[23..16]  | D[31..24]       |
| 0000 0010              | 0001 1011  | 1110 1110  | 0000 0001       |

### 8.4.15 Partial Reconfiguration Design Debugging

Use the Intel Quartus Prime software on-chip debugging tools, such as the Signal Tap logic analyzer, In-System Sources and Probes Editor, In-System Memory Content Editor, or JTAG Avalon Master Bridge to verify your partial reconfiguration design.

*Note:* Only the Signal Tap logic analyzer allows debugging of both the static and PR regions. Other tools support debugging only in the static region.

#### Related Links

[System Debugging Tools Overview](#)

In *Intel Quartus Prime Pro Edition Handbook Volume 3*

#### 8.4.15.1 Debugging a Partial Reconfiguration Design with Signal Tap Logic Analyzer

Unlike other debugging tools, Signal Tap Logic Analyzer uses the hierarchical debug capabilities provided by Intel Quartus Prime software. This feature allows you to tap signals in the static and PR regions simultaneously.

You can debug multiple personas present in your PR region, as well as multiple PR regions. For complete information on the debug infrastructure using hierarchical hubs, refer to *Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer* in the Intel Quartus Prime Pro Edition handbook.

#### Related Links

[Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer](#)

### 8.4.16 Partial Reconfiguration Simulation and Verification

Simulation verifies the behavior of your design before device programming. The Intel Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the PR control block. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

Similar to non-PR design simulations, preparing for a PR simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation. The Intel Quartus Prime software provides simulation components to help simulate a PR design, and can generate the gate-level PR simulation models for each persona. Use either the behavioral RTL or the gate level PR simulation model for simulation of the PR personas. The gate-level PR simulation model allows for accurate simulation of registers in your design, and reset sequence verification. These technology-mapped registers do not assume initial conditions.



## Related Links

[Simulating Intel FPGA Designs](#)

### 8.4.16.1 Partial Reconfiguration Simulation Flow

At a high-level, a PR operation consists of the following steps:

1. System-level preparation for a PR event.
2. Retrieval of the partial bitstream from memory.
3. Transmission of the partial bitstream to the Intel Arria 10 PR control block or Intel Stratix 10 SDM.
4. Resulting change in the design as a new persona becomes active.
5. Post-PR system coordination.
6. Use of the new persona in the system.

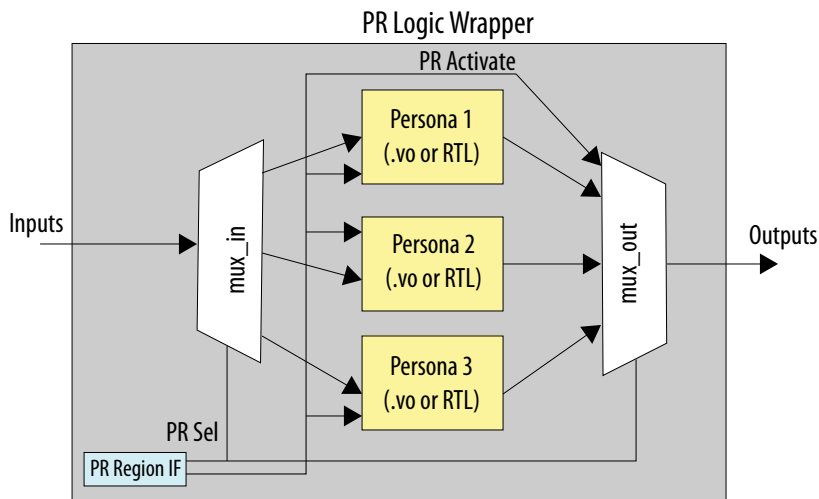
You can simulate each of these process steps in isolation, or as a larger sequence depending on your verification type requirement.

#### 8.4.16.1.1 Simulating PR Persona Replacement

The logical operation of the PR partition changes when a new persona loads during the partial reconfiguration process. Simulate the replacement of personas using multiplexers on the input and output of the persona under simulation. Create RTL wrapper logic to represent the top-level of the persona. The wrapper instantiates the default persona during compilation. During simulation, the wrapper allows the replacement of the active persona with another persona. Instantiate each persona as the behavioral RTL in the PR simulation model the Intel Quartus Prime EDA Netlist Writer generates. The Intel Quartus Prime software includes simulation modules to interface with your simulation testbench:

- `altera_pr_wrapper_mux_in`
- `altera_pr_wrapper_mux_out`
- `altera_pr_persona_if` (SystemVerilog interface allows you to connect the wrapper multiplexers to a testbench driver)

Figure 146. Simulation of PR Persona Switching



Example 74. RTL Wrapper for PR Persona Switching Simulation

The `pr_activate` input of the `altera_pr_wrapper_mux_out` module enables the MUX to output X. This functionality allows the simulation of unknown outputs from the PR persona, and also verifies the normal operation of the design's freeze logic. The following code corresponds to the simulation of PR persona switching, shown in the above figure:

```

module pr_core_wrapper
(
  input wire a,
  input wire b,
  output wire o
);

localparam ENABLE_PERSONA_1 = 1;
localparam ENABLE_PERSONA_2 = 1;
localparam ENABLE_PERSONA_3 = 1;
localparam NUM_PERSONA = 3;

logic pr_activate;
int persona_select;

altera_pr_persona_if persona_bfm();
assign pr_activate = persona_bfm.pr_activate;
assign persona_select = persona_bfm.persona_select;

wire a_mux [NUM_PERSONA-1:0];
wire b_mux [NUM_PERSONA-1:0];
wire o_mux [NUM_PERSONA-1:0];

generate
  if (ENABLE_PERSONA_1) begin
    localparam persona_id = 0;

    `ifndef ALTERA_ENABLE_PR_MODEL
      assign u_persona_0.altera_sim_pr_activate = pr_activate;
    `endif

    pr_and u_persona_0
    (
      .a(a_mux[persona_id]),
      .b(b_mux[persona_id]),

```

```

        .o(o_mux[persona_id])
    );
    end
endgenerate

generate
    if (ENABLE_PERSONA_2) begin
        localparam persona_id = 1;

        `ifdef ALTERA_ENABLE_PR_MODEL
            assign u_persona_1.altera_sim_pr_activate = pr_activate;
        `endif

        pr_or u_persona_1
        (
            .a(a_mux[persona_id]),
            .b(b_mux[persona_id]),
            .o(o_mux[persona_id])
        );

    end
endgenerate

generate
    if (ENABLE_PERSONA_3) begin
        localparam persona_id = 2;

        `ifdef ALTERA_ENABLE_PR_MODEL
            assign u_persona_2.altera_sim_pr_activate = pr_activate;
        `endif

        pr_empty u_persona_2
        (
            .a(a_mux[persona_id]),
            .b(b_mux[persona_id]),
            .o(o_mux[persona_id])
        );

    end
endgenerate

altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1)) /
    u_a_mux(.sel(persona_select), .mux_in(a), .mux_out(a_mux));

altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1)) /
    u_b_mux(.sel(persona_select), .mux_in(b), .mux_out(b_mux));

altera_pr_wrapper_mux_out #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1)) /
    u_o_mux(.sel(persona_select), .mux_in(o_mux), .mux_out(o), .pr_activate /
    (pr_activate));

endmodule

```

#### 8.4.16.1.2 PR Simulation Wrapper Modules

##### altera\_pr\_persona\_if Module

Instantiate the `altera_pr_persona_if` SystemVerilog interface in a PR region simulation wrapper to connect to all the wrapper multiplexers. Optionally, connect `pr_activate` to the PR simulation model.





Connect the interface's `persona_select` to the `sel` port of all input and output multiplexers. Connect the `pr_activate` to the `pr_activate` of all the output multiplexers. Optionally, connect the report events to the report event ports of the PR simulation model. Then, the PR region driver testbench component can drive the interface.

```
interface altera_pr_persona_if;
    logic pr_activate;
    int    persona_select;

    event report_storage_if_x_event;
    event report_storage_if_l_event;
    event report_storage_if_0_event;
    event report_storage_event;

    initial begin
        pr_activate <= 1'b0;
    end
endinterface : altera_pr_persona_if
```

The `<QUARTUS_INSTALL_DIR>/eda/sim_lib/altera_lnsim.sv` file defines the `altera_pr_persona_if` component.

### altera\_pr\_wrapper\_mux\_out Module

The `altera_pr_wrapper_mux_out` module allows you to multiplex the outputs of all PR personas to the outputs of the PR region wrapper.

Instantiate one multiplexer per output port. Specify the active persona using the `sel` port of the multiplexer. The `pr_activate` port allows you to drive the multiplexer output to "x", to emulate the unknown value of PR region outputs during a PR operation. Parameterize the component to specify the number of persona inputs, the multiplexer width, and the MUX output value when `pr_activate` asserts.

```
module altera_pr_wrapper_mux_out #(
    parameter NUM_PERSONA = 1,
    parameter WIDTH = 1,
    parameter [0:0] DISABLED_OUTPUT_VAL = 1'bx
) (
    input int sel,
    input wire [WIDTH-1 : 0] mux_in [NUM_PERSONA-1:0],
    output reg [WIDTH-1:0] mux_out,
    input wire pr_activate
);

always_comb begin
    if ((sel < NUM_PERSONA) && (!pr_activate))
        mux_out = mux_in[sel];
    else
        mux_out = {WIDTH{DISABLED_OUTPUT_VAL}};
    end
endmodule : altera_pr_wrapper_mux_out
```

The `<QUARTUS_INSTALL_DIR>/eda/sim_lib/altera_lnsim.sv` file defines the `altera_pr_wrapper_mux_out` component.

### altera\_pr\_wrapper\_mux\_in Module

The `altera_pr_wrapper_mux_in` module allows you to de-multiplex inputs to a PR partition wrapper for all PR personas.

Instantiate one multiplexer per input port. Specify the active persona using the `sel` port of the multiplexer. Parameterize the component to specify the number of persona outputs, the multiplexer width, and the MUX output for any disabled output. When

using the `altera_pr_wrapper_mux_in` to mux a clock input, use the `DISABLED_OUTPUT_VAL` of 0, to ensure there are no simulation clock events of the disabled personas.

```

module altera_pr_wrapper_mux_in#(
  parameter NUM_PERSONA = 1,
  parameter WIDTH = 1,
  parameter [0:0] DISABLED_OUTPUT_VAL = 1'bx
) (
  input int sel,
  input wire [WIDTH-1:0] mux_in,
  output reg [WIDTH-1 : 0] mux_out [NUM_PERSONA-1:0]
);
always_comb begin
  for (int i = 0; i < NUM_PERSONA; i++)
    if (i == sel)
      mux_out[i] = mux_in;
    else
      mux_out[i] = {WIDTH{DISABLED_OUTPUT_VAL}};
end
endmodule : altera_pr_wrapper_mux_in

```

The `<QUARTUS_INSTALL_DIR>/eda/sim_lib/altera_lnsim.sv` file defines the `altera_pr_wrapper_mux_in` component.

#### 8.4.16.1.3 Generating the PR Persona Simulation Model

Use the Intel Quartus Prime EDA Netlist Writer to create the simulation model for a PR persona. The simulation model represents the post-synthesis, gate-level netlist for the persona.

When using the PR simulation model for the persona, the netlist includes a new `altera_sim_pr_activate` top-level signal for the model. You can asynchronously drive this signal to load all registers in the model with X. This feature allows you to verify the reset sequence of the new persona on PR event completion. Verify the reset sequence through inspection, using SystemVerilog assertions, or using other checkers.

By default, the PR simulation model asynchronously loads X into the register's storage element on `pr_activate` signal assertion. You can parameterize this behavior on a per register basis, or on a simulation-wide default basis. The simulation model supports four built-in modes:

- load X
- load 1
- load 0
- load rand

Specify these modes using the SystemVerilog classes:

- `dffeas_pr_load_x`
- `dffeas_load_1`
- `dffeas_load_0`
- `dffeas_load_rand`

Optionally, you can create your own PR activation class, where your class must define the `pr_load` variable to specify the PR activation value.



Follow these steps to generate the simulation model for a PR design:

1. To run synthesis and generate the simulation netlist for your EDA simulator, click **Synthesis** on the Compilation Dashboard.

*Note:* The current version of the Intel Quartus Prime software supports the PR simulation model only in SystemVerilog.

2. To generate the PR simulation model, type the following from the command-line:

```
quartus_eda --pr --simulation --tool={your_tool} project -c pr_syn_revision
```

3. To specify a simulation-wide behavior, set the `ALTERA_DEFAULT_DFFEAS_PR_ACTIVATE_CLASS` macro to the name of the class to use for initialization. For example:

```
define ALTERA_DEFAULT_DFFEAS_PR_ACTIVATE_CLASS dffeas_pr_load_1
```

4. To specify the behavior for an individual register, set the `PR_ACTIVATE_CLASS` parameter of the specific `dffeas_pr` register to the desired initialization class. For more information, refer to the `dffeas_pr` model in the `altera_lnsim.sv` file, located in `<QUARTUS_INSTALL_DIR>/eda/sim_lib/altera_lnsim.sv`.

*Note:* The Aldec Riviera-PRO\* Simulator does not support selecting different `PR_ACTIVATE_CLASS` parameters, and only supports registers going to X during `pr_activate`.

#### Example 75. Built-in Initialization Classes

```
class dffeas_pr_load_x;  
    reg pr_load = 1'bx;  
  
    function new();  
    endfunction  
  
endclass  
  
class dffeas_pr_load_0;  
    reg pr_load = 1'b0;  
  
    function new();  
    endfunction  
  
endclass  
  
class dffeas_pr_load_1;  
    reg pr_load = 1'b1;  
  
    function new();  
    endfunction  
  
endclass  
  
class dffeas_pr_load_rand;  
    rand bit pr_load;  
  
    function new(int seed = $random());  
        this.srandom(seed);  
    endfunction  
  
endclass
```

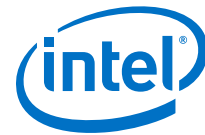
## 8.5 Partial Reconfiguration Design Recommendations

When designing for partial reconfiguration, consider the system-level behavior to maintain the integrity and correctness of the static region operation. For example, during PR programming, ensure that the system does not read or write to the PR region. In addition, freeze the write enable output from the PR region into the static region. This freezing avoids interference with the static region operation.

**Table 87. Partial Reconfiguration Design Guidelines**

| Scenario                                                              | Guideline                                                                                                                                                                         | Reasoning                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Designing for partial reconfiguration                                 | Do not assume initial states in registers. Ensure that you reset all the registers.                                                                                               | The registers contain undefined values after reconfiguration.                                                                                                                                                                                                                                                                                                                                                                           |
|                                                                       | Reset the registers that drive control signals to a known state, after partial reconfiguration.                                                                                   | Registers contain undefined values after reconfiguration. In addition, synthesis can duplicate registers.                                                                                                                                                                                                                                                                                                                               |
|                                                                       | You cannot define synchronous reset as a global signal for Intel Arria 10 partial reconfiguration.                                                                                | PR regions do not support synchronous reset of registers as a global signal, because the Intel Arria 10 LAB does not support synchronous clear ( <code>sclr</code> ) signal on a global buffer. The LAB supports the asynchronous clear ( <code>acclr</code> ) signal driven from a local input, or from a global network row clock. As a result, only the <code>acclr</code> can be a global signal, feeding registers in a PR region. |
| Partitioning the design                                               | Register all the inputs and outputs for your PR region.                                                                                                                           | Improves timing closure and time budgeting.                                                                                                                                                                                                                                                                                                                                                                                             |
|                                                                       | Reduce the number of signals interfacing the PR region with the static region in your design.                                                                                     | Reduces the wire LUT count.                                                                                                                                                                                                                                                                                                                                                                                                             |
|                                                                       | Create a wrapper for your PR region.                                                                                                                                              | The wrapper creates common footprint to static region.                                                                                                                                                                                                                                                                                                                                                                                  |
|                                                                       | Drive all the PR region output ports to inactive state.                                                                                                                           | Prevents the static region logic from receiving random data during the partial reconfiguration operation.                                                                                                                                                                                                                                                                                                                               |
|                                                                       | PR boundary I/O interface must be a superset of all the PR persona I/O interfaces.                                                                                                | Ensures that each PR partition implements the same ports.                                                                                                                                                                                                                                                                                                                                                                               |
| Preparing for partial reconfiguration                                 | Complete all pending transactions.                                                                                                                                                | Ensures that the static region is not in a wait state.                                                                                                                                                                                                                                                                                                                                                                                  |
| Maintaining a partially working system during partial reconfiguration | Hold all outputs to known constant values.                                                                                                                                        | Ensures that the undefined values the PR region receives during and after the reconfiguration do not affect the PR control logic.                                                                                                                                                                                                                                                                                                       |
| Initializing after partial reconfiguration                            | Initialize after reset.                                                                                                                                                           | Retrieves state from memory or other device resources.                                                                                                                                                                                                                                                                                                                                                                                  |
| Debugging the design using Signal Tap Logic Analyzer                  | <ul style="list-style-type: none"> <li>Do not tap signals in the default personas.</li> <li>Store all the tapped signals from a persona in one <code>.stp</code> file.</li> </ul> | The current version of the Intel Quartus Prime software supports only one <code>.stp</code> (Signal Tap file) per revision. This limitation requires you to select partitions, one at a time, to tap.                                                                                                                                                                                                                                   |

*continued...*



| Scenario | Guideline                                                                                             | Reasoning                                                            |
|----------|-------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
|          | Do not tap across regions in the same .stp file.                                                      | Ensures consistent interface (boundary) across all personas.         |
|          | Tap only the pre-synthesis signals. In the Node Finder, filter for <b>Signal Tap: pre-synthesis</b> . | Ensures that the signal tapping of PR personas start from synthesis. |
|          | Only include the .stp file in the PR synthesis revision.                                              | Supports only pre-synthesis tapping.                                 |

## 8.6 Partial Reconfiguration Design Considerations

Partial reconfiguration is an advanced design flow in the Intel Quartus Prime Pro Edition software. Successfully creating a partial reconfiguration design requires understanding the requirements and best design practices for the PR flow.

The following list summarizes the design considerations for partial reconfiguration:

- Reconfigurable partitions can only contain core resources, such as LABs, RAMs, and DSPs. All periphery resources, such as the transceivers, external memory interface, HPS, and clocks must be in the static portion of the design.
- To physically partition the device between static and individual PR regions, floorplan each PR region into exclusive, core-only, placement regions, with associated routing regions.
- A reconfiguration partition must contain the super-set of all ports that you use across all PR personas.
- To minimize Intel Arria 10 programming files size, ensure that the PR regions are short and wide. For Intel Stratix 10 designs, use sector-aligned PR regions.
- The maximum number of clocks or other global signals for any Intel Arria 10 PR region is 33. The maximum number of clocks or other global signals for any Intel Stratix 10 PR region is 32. In the current version of the Intel Quartus Prime Pro Edition software, no two PR regions can share a row-clock.
- PR regions do not require any input freeze logic. However, you must freeze all the outputs of each PR region to a known constant value to avoid unknown data during partial reconfiguration.
- Your PR design must consider all the system-level coordination of partial reconfiguration.
- The current version of the Intel Quartus Prime Pro Edition software supports only one .stp (signal tap file) per revision. Only include the .stp file in the PR synthesis revision.
- Increase the reset length by 1 cycle to account for register duplication in the Fitter.
- Only Intel Arria 10 devices in -1, -2 and -3 speed grade support partial reconfiguration. All Intel Stratix 10 devices support PR.
- Use the nominal VCC of 0.9V or 0.95V as per the datasheet, including VID enabled devices.
- Intel Quartus Prime Standard Edition software does not support partial reconfiguration for Intel Arria 10 devices or Intel Stratix 10 devices.



## 8.7 Document Revision History

This document has the following revision history.

**Table 88. Document Revision History**

| Date       | Version | Changes                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2017.11.06 | 17.1.0  | <ul style="list-style-type: none"><li>Added partial reconfiguration support for Intel Stratix 10 devices.</li><li>Added descriptions of Intel Stratix 10 Partial Reconfiguration Controller IP, SUPR, HPR, and SDM to terms list.</li><li>Updated for latest Intel branding and software user interface.</li></ul>                                                                                                             |
| 2017.05.08 | 17.0.0  | <ul style="list-style-type: none"><li>Added information about Hierarchical Partial Reconfiguration.</li><li>Added new topic Partial Reconfiguration Simulation and Verification.</li><li>Added new topic 'Run Timing Analysis on a Design with Multiple PR Partitions'.</li><li>Updated Freeze Logic for PR Regions.</li><li>Added new topic Debugging Using Signal Tap Logic Analyzer.</li><li>Other minor updates.</li></ul> |
| 10.31.2016 | 16.1.0  | <ul style="list-style-type: none"><li>Implemented Intel rebranding.</li><li>Initial release.</li></ul>                                                                                                                                                                                                                                                                                                                         |

### Related Links

#### [Altera Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the Altera documentation archives.



## 9 Creating a System with Platform Designer

---

Platform Designer is a system integration tool included as part of the Intel Quartus Prime software.

*Note:* Intel now refers to Qsys Pro as Platform Designer.

Platform Designer simplifies the task of defining and integrating customized IP Components (IP Cores) into your designs. Platform Designer simplifies the task of defining and integrating customized IP Components (IP Cores) into your designs.

Platform Designer facilitates design reuse by packaging and integrating your custom IP components with Intel and third-party IP components. Platform Designer automatically creates interconnect logic from the high-level connectivity that you specify, which eliminates the error-prone and time-consuming task of writing HDL to specify system-level connections.

Platform Designer introduces hierarchical isolation between system interconnect and IP components. Platform Designer stores the instantiated IP component in a separate `.ip` file and the system connectivity information in the `.qsys` file. This hierarchical isolation ensures that changing the parameters of a single IP component does not necessitate regeneration of the enclosing system or any other IP component within that system. Likewise, a change to system connectivity does not require regeneration of any of the IP components. Platform Designer references the parameterized IP component for instantiation in the system by the component's entity name, and generates the RTL of the IP component and the RTL of the system separately.

Platform Designer is a more powerful tool if you design your custom IP components using standard interfaces available in the Platform Designer IP Catalog. Standard interfaces inter-operate efficiently with the Intel FPGA IP components, and you can take advantage of bus functional models (BFMs), monitors, and other verification IP to verify your systems.

Platform Designer supports Avalon, Arm\* AMBA\* 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

Platform Designer provides the following advantages:

- Simplifies the process of customizing and integrating IP components into systems
- Provides isolation between the system and IP component, maintaining all the parameter information of the IP component in a separate `.ip` file.
- Supports generic components, allowing the instantiation of IP components without an HDL implementation.
- Generates an IP core variation for use in your Intel Quartus Prime software projects



- Supports incremental generation of the system and IP components.
- Allows specifying interface requirements for the system.
- Supports up to 64-bit addressing
- Supports modular system design
- Supports visualization of systems
- Supports optimization of interconnect and pipelining within the system
- Supports auto-adaptation of different data widths and burst characteristics
- Supports inter-operation between standard protocols
- Fully integrated with the Intel Quartus Prime software

**Note:** For information on how to define and generate stand-alone IP cores for use in your Intel Quartus Prime software projects, refer to *Introduction to Intel FPGA IP Cores* and *Managing Intel Quartus Prime Projects*.

#### Related Links

- [Introduction to Intel FPGA IP Cores](#)
- [Managing Intel Quartus Prime Projects](#) on page 32
- [Avalon Interface Specifications](#)
- [AMBA Protocol Specifications](#)

## 9.1 Interface Support in Platform Designer

IP components (IP Cores) can have any number of interfaces in any combination. Each interface represents a set of signals that you can connect within a Platform Designer system, or export outside of a Platform Designer system.

Platform Designer IP components can include the following interface types:

**Table 89. IP Component Interface Types**

| Interface Type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Memory-Mapped  | Connects memory-referencing master devices with slave memory devices. Master devices may be processors and DMAs, while slave memory devices may be RAMs, ROMs, and control registers. Data transfers between master and slave may be uni-directional (read only or write only), or bi-directional (read and write).                                                                                                                                                                                                  |
| Streaming      | Connects Avalon Streaming (Avalon-ST) sources and sinks that stream unidirectional data, as well as high-bandwidth, low-latency IP components. Streaming creates datapaths for unidirectional traffic, including multichannel streams, packets, and DSP data. The Avalon-ST interconnect is flexible and can implement on-chip interfaces for industry standard telecommunications and data communications cores, such as Ethernet, Interlaken, and video. You can define bus widths, packets, and error conditions. |
| Interrupts     | Connects interrupt senders to interrupt receivers. Platform Designer supports individual, single-bit interrupt requests (IRQs). In the event that multiple senders assert their IRQs simultaneously, the receiver logic (typically under software control) determines which IRQ has highest priority, then responds appropriately                                                                                                                                                                                    |

*continued...*





| Interface Type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Clocks         | Connects clock output interfaces with clock input interfaces. Clock outputs can fan-out without the use of a bridge. A bridge is required only when a clock from an external (exported) source connects internally to more than one source.                                                                                                                                                                                                                                                                                                                                                                             |
| Resets         | Connects reset sources with reset input interfaces. If your system requires a particular positive-edge or negative-edge synchronized reset, Platform Designer inserts a reset controller to create the appropriate reset signal. If you design a system with multiple reset inputs, the reset controller ORs all reset inputs and generates a single reset output.                                                                                                                                                                                                                                                      |
| Conduits       | Connects point-to-point conduit interfaces, or represent signals that are exported from the Platform Designer system. Platform Designer uses conduits for component I/O signals that are not part of any supported standard interface. You can connect two conduits directly within a Platform Designer system as a point-to-point connection, or conduit interfaces can be exported and brought to the top-level of the system as top-level system I/O. You can use conduits to connect to external devices, for example external DDR SDRAM memory, and to FPGA logic defined outside of the Platform Designer system. |

## 9.2 Introduction to the Platform Designer IP Catalog

The Platform Designer IP Catalog offers a broad range of configurable IP Cores optimized for Intel devices to use in your Platform Designer designs.

The Intel Quartus Prime software installation includes the Intel FPGA IP library. You can integrate optimized and verified Intel FPGA IP cores into your design to shorten design cycles and maximize performance. The IP Catalog can include Intel-provided IP components, third-party IP components, custom IP components that you create in the Platform Designer Component Editor, and previously generated Platform Designer systems.

The Platform Designer IP Catalog includes the following IP component types:

- Microprocessors, such as the Nios II processor
- DSP IP cores, such as the Reed Solomon Decoder II
- Interface protocols, such as the IP Compiler for PCI Express
- Memory controllers, such as the RLDRAM II Controller with UniPHY
- Avalon Streaming (Avalon-ST) IP cores, such as the Avalon-ST Multiplexer
- Platform Designer Interconnect
- Verification IP (VIP) Bus Functional Models (BFMs)

### Related Links

[Introduction to Intel FPGA IP Cores](#)

### 9.2.1 Installing and Licensing IP Cores

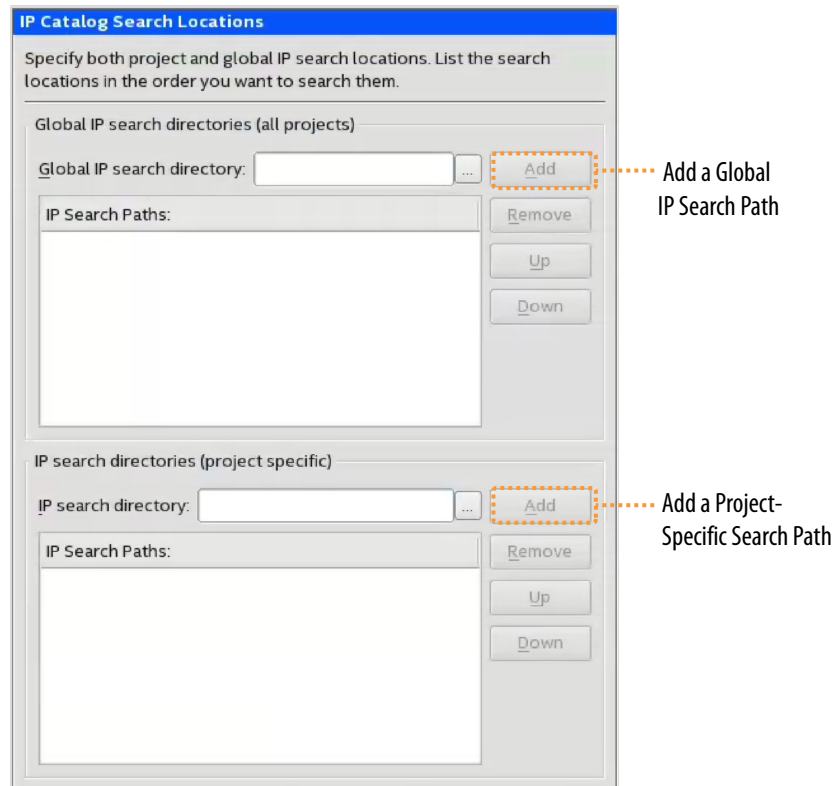
The Intel Quartus Prime software includes the Intel FPGA IP Library. The library provides many useful IP core functions for production use without additional license. You can fully evaluate any licensed Intel FPGA IP core in simulation and in hardware until you are satisfied with its functionality and performance. The HDMI IP core is part of the Intel FPGA IP Library, which is distributed with the Intel Quartus Prime software and downloadable from [www.altera.com](http://www.altera.com).

After you purchase a license for the IP core, you can request a license file from the licensing site and install it on your computer. When you request a license file, Intel emails you a `license.dat` file. If you do not have Internet access, contact your local Intel representative.

### 9.2.2 Adding IP Cores to IP Catalog

The IP Catalog automatically displays IP cores located in the project directory, in the default Intel Quartus Prime installation directory, and in the IP search path.

**Figure 147. Specifying IP Search Locations**



The IP Catalog displays Intel Quartus Prime IP components and Platform Designer systems, third-party IP components, and any custom IP components that you include in the path. Use the **IP Search Path** option (**Tools > Options**) to include custom and third-party IP components in the IP Catalog.

The Intel Quartus Prime software searches the directories listed in the IP search path for the following IP core files:

- Component Description File (`_hw.tcl`)—defines a single IP core.
- IP Index File (`.ipx`)—each `.ipx` file indexes a collection of available IP cores. This file specifies the relative path of directories to search for IP cores. In general, `.ipx` files facilitate faster searches.



The Intel Quartus Prime software searches some directories recursively and other directories only to a specific depth. When the search is recursive, the search stops at any directory that contains a `_hw.tcl` or `.ipx` file.

In the following list of search locations, `**` indicates a recursive descent.

**Table 90. IP Search Locations**

| Location            | Description                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------|
| PROJECT_DIR/*       | Finds IP components and index files in the Intel Quartus Prime project directory.                                             |
| PROJECT_DIR/ip/**/* | Finds IP components and index files in any subdirectory of the /ip subdirectory of the Intel Quartus Prime project directory. |

If the Intel Quartus Prime software recognizes two IP cores with the same name, the following search path precedence rules determine the resolution of files:

1. Project directory.
2. Project database directory.
3. Project IP search path specified in **IP Search Locations**, or with the `SEARCH_PATH` assignment for the current project revision.
4. Global IP search path specified in **IP Search Locations**, or with the `SEARCH_PATH` assignment in the `quartus2.ini` file.
5. Quartus software libraries directory, such as `<Quartus Installation>\libraries`.

*Note:* If you add an IP component to the search path, update the IP Catalog by clicking **Refresh IP Catalog** in the drop-down list. In Platform Designer and Platform Designer, click **File > Refresh System** to update the IP Catalog.

### 9.2.3 General Settings for IP

Use the following settings to control how the Intel Quartus Prime software manages IP cores in your project.

**Table 91. IP Core General Setting Locations**

| Setting Location                                                                                                                                        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Tools &gt; Options &gt; IP Settings</b><br>Or<br><b>Tasks pane &gt; Settings &gt; IP Settings</b><br>(Pro Edition Only)                              | <ul style="list-style-type: none"> <li>• Specify the <b>IP generation HDL preference</b>. The parameter editor generates the HDL you specify for IP variations.</li> <li>• Increase the <b>Maximum Platform Designer memory usage size</b> if you experience slow processing for large systems, or for out of memory errors.</li> <li>• Specify whether to <b>Automatically add Intel Quartus Prime IP files</b> to all projects. Disable this option to manually add the IP files.</li> <li>• Use the <b>IP Regeneration Policy</b> setting to control when synthesis files regenerate for each IP variation. Typically, you <b>Always regenerate synthesis files for IP cores</b> after making changes to an IP variation.</li> </ul> |
| <b>Tools &gt; Options &gt; IP Catalog Search Locations</b><br>Or<br><b>Tasks pane &gt; Settings &gt; IP Catalog Search Locations</b> (Pro Edition Only) | <ul style="list-style-type: none"> <li>• Specify additional project and global IP search locations. The Intel Quartus Prime software searches for IP cores in the project directory, in the Intel Quartus Prime installation directory, and in the IP search path.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## 9.2.4 Set up the IP Index File (.ipx) to Search for IP Components

An IP Index File (.**ipx**) contains a search path that Platform Designer uses to search for IP components. You can use the `ip-make-ipx` command to create an **.ipx** file for any directory tree, which can reduce the startup time for Platform Designer.

You can specify a search path in the **user\_components.ipx** file in either in the Intel Quartus Prime software (**Tools > Options > IP Catalog Search Locations**). This method of discovering IP components allows you to add a locations dependent of the default search path. The **user\_components.ipx** file directs Platform Designer to the location of an IP component or directory to search.

A `<path>` element in the **.ipx** file specifies a directory where multiple IP components may be found. A `<component>` entry specifies the path to a single component. A `<path>` element can use wildcards in its definition. An asterisk matches any file name. If you use an asterisk as a directory name, it matches any number of subdirectories.

### Example 76. Path Element in an .ipx File

```
<library>
  <path path="...<user directory>" />
  <path path="...<user directory>" />
  ...
  <component ... file="...<user directory>" />
  ...
</library>
```

A `<component>` element in an **.ipx** file contains several attributes to define a component. If you provide the required details for each component in an **.ipx** file, the startup time for Platform Designer is less than if Platform Designer must discover the files in a directory. The example below shows two `<component>` elements. Note that the paths for file names are specified relative to the **.ipx** file.

### Example 77. Component Element in an .ipx File

```
<library>
  <component
    name="A Platform Designer Component"
    displayName="Platform Designer FIR Filter Component"
    version="2.1"
    file="./components/qsys_filters/fir_hw.tcl"
  />
  <component
    name="rgb2cmky_component"
    displayName="RGB2CMYK Converter(Color Conversion Category!)"
    version="0.9"
    file="./components/qsys_converters/color/rgb2cmky_hw.tcl"
  />
</library>
```

**Note:** You can verify that IP components are available with the `ip-catalog` command.

### Related Links

[Create an .ipx File with ip-make-ipx on page 398](#)



## 9.2.5 Integrate Third-Party IP Components into the Platform Designer IP Catalog

You can use IP components created by Intel partners in your Platform Designer systems. These IP components have interfaces that are supported by Platform Designer, such as Avalon-MM or AXI. Additionally, some include timing and placement constraints, software drivers, simulation models, and reference designs.

To locate supported third-party IP components on Altera's web page, navigate to the *Intellectual Property & Reference Designs* page, type `Platform Designer Certified` in the **Search** box, select **IP Core & Reference Designs**, and then press **Enter**.

Refer to Intel's *Intellectual Property & Reference Designs* page for more information.

### Related Links

[Intellectual Property & Reference Designs](#)

## 9.3 Create a Platform Designer System

Click **Tools** > **Platform Designer** in the Intel Quartus Prime software to open Platform Designer. A `.qsys` file represents your Platform Designer system in your Intel Quartus Prime software project.


### Related Links

- [Creating Platform Designer Components](#) on page 608
- [Component Interface Tcl Reference](#) on page 791

### 9.3.1 Create/Open Project in Platform Designer

The Intel Quartus Prime software tightly links with Platform Designer system creation. Platform Designer requires you to specify a Intel Quartus Prime project at time of system creation.

To create a new system, or open an existing system in Platform Designer:

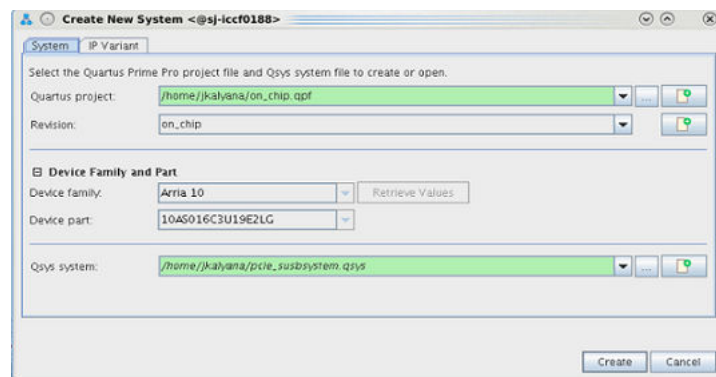
1. To create a new Intel Quartus Prime project to associate with your Platform Designer system, click . To select an existing project, browse for the project. Alternatively, select an existing project from the drop-down list in the **Quartus project** field.

*Note:* Selecting **None** from the drop-down list in the **Quartus project** field opens the Platform Designer tool in view-only mode.<sup>(8)</sup>

2. To create a new revision for the Intel Quartus Prime project, click . To specify an existing revision for the project, select an existing revision from the drop-down list in the **Revision** field.
3. When creating a new Intel Quartus Prime project, specify the **Device family** and **Device part** to associate with your Platform Designer system by selecting the device name and device part number from the respective fields. If you are opening an existing Intel Quartus Prime project to associate with your Platform Designer system, click **Retrieve Values** to populate the fields with the device information of the Intel Quartus Prime project.
4. To create a new Platform Designer system, click . To open an existing .qsys file, browse for the file. Alternatively, select an existing file from the drop-down list.

*Note:* Similarly, you can open an existing IP file, or create a new IP variant by selecting the **IP Variant** tab in the **Create New System** dialog box. To create a new IP variant, you must specify a **Component type** for the .ip file.

**Figure 148. Platform Designer Create New System**



*Note:*

- To change the Intel Quartus Prime project associated with your current Platform Designer system, click **File > Select Quartus Project**.

### 9.3.1.1 Convert your Existing System to Platform Designer Format

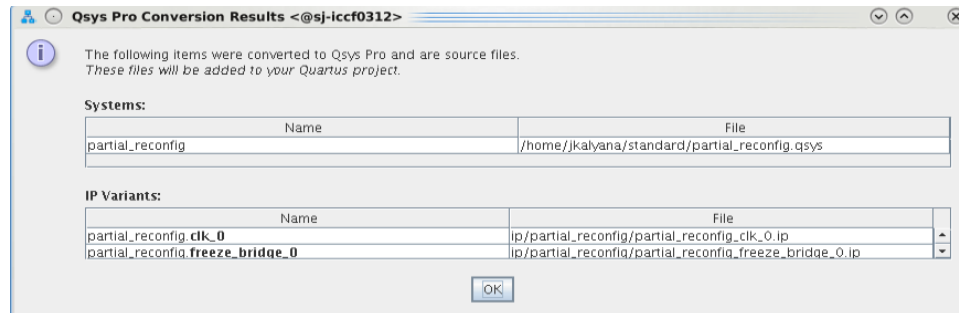
When you open an existing system with incompatible components, Platform Designer prompts you to convert these components to the Platform Designer format. On conversion, the **Platform Designer Conversion Results** dialog box appears, listing all the converted system and IP source files.

<sup>(8)</sup> View-only mode restricts the following functionality:

- Adding new IP components to the system or subsystem.
- Removing the instantiated IP components from the system or subsystem.
- Creating a new system, subsystem, or IP file.
- Executing system scripts.

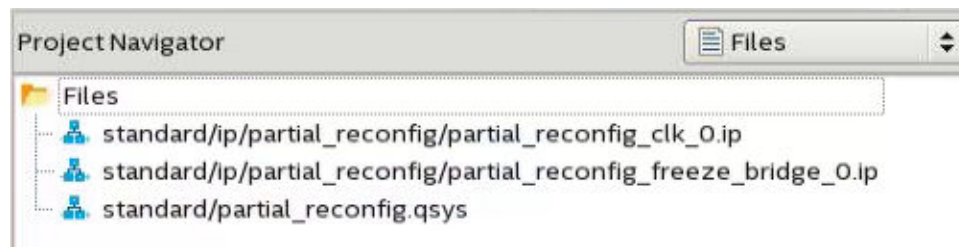


Figure 149. Platform Designer Conversion Results Dialog Box



Platform Designer stores the .ip files inside an ip folder, relative to the .qsys system file location. Platform Designer prefixes the system name to the .ip file name. Platform Designer automatically adds these converted files to the associated Intel Quartus Prime project. Ensure that you maintain these .ip files, along with your system files.

Figure 150. System and IP Files Associated with a Intel Quartus Prime Project



### 9.3.2 Modify the Target Device

The Platform Designer system inherits the device family from the associated Intel Quartus Prime project.

You can modify the device settings of your Platform Designer system from the **Device Family** tab. Changing the **Device family** or **Device** options from this tab automatically updates the associated Intel Quartus Prime project.

### 9.3.3 Modify the IP Search Path

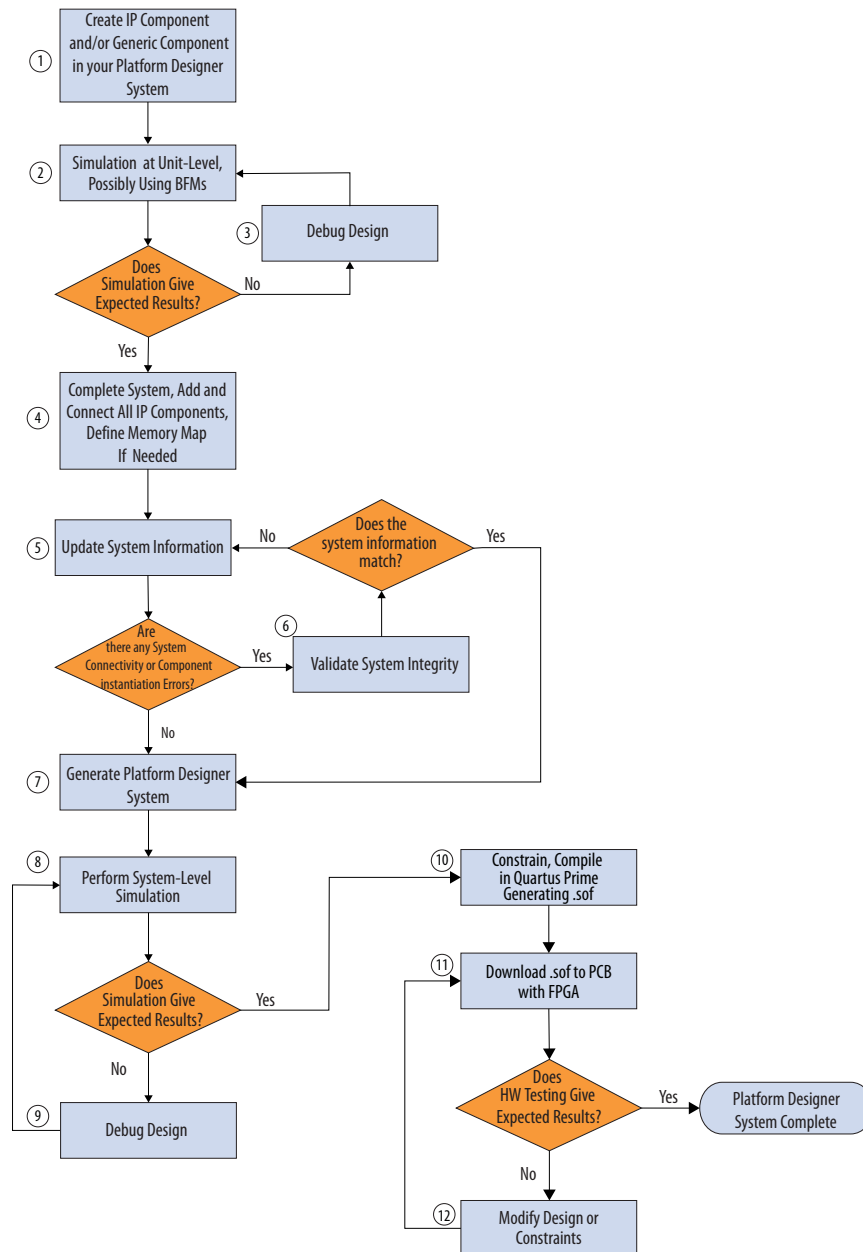
Platform Designer allows you to view and modify the IP search locations specified for the Intel Quartus Prime project associated with your system. To specify the IP search path from Platform Designer:

1. Click **Tools** > **Options** > **IP Search Path**. The Intel Quartus Prime Global IP Search Path and Quartus Project IP Search Path panes display the IP search locations specified for your associated Intel Quartus Prime project.
2. Click **Add** or **Remove** to add/remove new search locations. The Intel Quartus Prime project automatically updates to reflect these modifications. In Platform Designer, click **File** > **Refresh System** to propagate these changes.

### 9.3.4 Platform Designer System Design flow

The Platform Designer design flow involves creating, instantiating and generating, and simulating system output for IP components.

Figure 151. Platform Designer System Design Flow



**Note:** For information on how to define and generate single IP cores for use in your Intel Quartus Prime software projects, refer to *Introduction to Intel FPGA IP Cores*.





## Related Links

[Introduction to Intel FPGA IP Cores](#)

### 9.3.5 Add IP Components (IP Cores) to a Platform Designer System

The Platform Designer IP Catalog displays IP components (IP cores) available for your target device. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to create a custom IP component variation of the selected component. A Platform Designer system can contain a single instance of an IP component, or multiple, individually parameterized variations of multiple or the same IP components.

Platform Designer preserves each of the IP component's parameters as a `.ip` file. A Platform Designer system instantiates a generic component in place of the actual IP core with a reference to the HDL entity name, module and interface assignments, compilation library, HDL ports, interfaces, and system-info parameters.

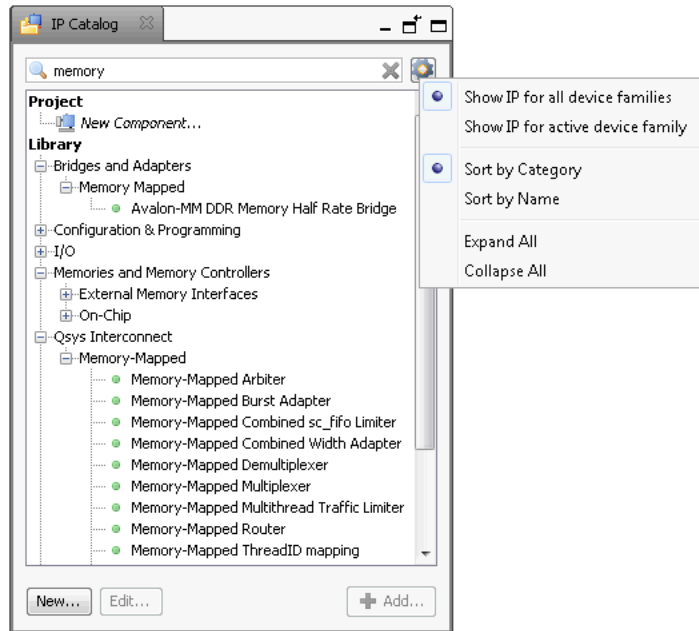
Follow these steps to locate, instantiate, and customize an IP component in your Platform Designer system:

1. Right-click any IP component name in the Platform Designer IP Catalog to display details about device support, installation location, versions, and links to documentation.
2. To locate a specific type of component, type some or all of the component's name in the **IP Catalog** search box.  
For example, type **memory** to locate memory-mapped IP components, or **axi** to locate AXI IP. You can also filter the IP Catalog display with options on the right-click menu.
3. To launch the parameter editor, double-click any component. You can set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in the parameterization of the IP component.
4. For IP components that have preset parameter values, select the preset file in the preset editor, and then click **Apply**. This option allows you to instantly apply preset parameter values for the IP component appropriate for a specific application.
5. To complete customization of the IP component, click **Finish**. The IP component appears in the **System Contents** and **Component Instantiation** tabs.

*Note:* Platform Designer creates a corresponding `.ip` file for the IP component on instantiation, and stores the file in the `<ip>` folder in your project directory.

The IP component appears in the **System Contents** tab.

Figure 152. Platform Designer IP Catalog



### 9.3.6 Specify Implementation Type for IP Components

A Platform Designer system instantiates a generic component in place of the actual IP core with a reference to the HDL entity name, module and interface assignments, compilation library, HDL ports, interfaces, and system-info parameters.

The **Component Instantiation** tab allows you to configure the system representation of an IP core. To open the **Component Instantiation** tab, click **View > Component Instantiation**.

Table 92. Component Instantiation GUI Information

Name	Description
<b>Implementation Type</b>	Allows you to decide how to define the implementation of your IP component. Platform Designer has the following implementation types:
<i>continued...</i>	



Name	Description
	<ul style="list-style-type: none"> <li>• <b>IP</b>—The default implementation type for any IP core. With <b>IP Implementation Type</b>, Platform Designer performs the following functions:               <ul style="list-style-type: none"> <li>— Runs background checks against the port widths between the IP component and the <code>.ip</code> file to ensure continuity.</li> <li>— Scans the <code>.ip</code> file for the error flag to understand if any component has parameterization errors.</li> <li>— Checks for system-info mismatches between the IP file and the IP component in the system and prompts you to resolve these through IP instantiation warnings in the <b>Instantiation Messages</b> tab.</li> </ul> </li> <li>• <b>HDL</b>—Allows you to quickly import RTL to your Platform Designer system. You can populate the signals and interfaces parameters of the generic component from an RTL file.</li> <li>• <b>Blackbox</b>—By choosing this implementation type, you specify a component that represents the signal and interface boundary of an entity, without providing the component's implementation. You must provide the implementation of the component in a downstream compiler such as Intel Quartus Prime software or your RTL simulator.</li> <li>• <b>HLS</b>—Select to add an existing high level synthesis (HLS) file, compile an HLS file, import a previously compiled HLS file, perform verification on an HLS project, or display the resulting compilation report.</li> </ul>
<b>Compilation Info</b>	Allows you to specify the <b>HDL Entity name</b> and <b>HDL compilation library</b> name for the implementation. These are fixed values for the <b>IP Implementation Type</b> .
<b>Signals &amp; Interfaces</b>	Allows you to define the port boundary of the component. Click <b>&lt;&lt;add interface&gt;&gt;</b> or <b>&lt;&lt;add signal&gt;&gt;</b> to add the interfaces and signals.
<b>System Information</b>	Allows you to specify the address map of the interfaces, input clock rate, and other necessary system information associated with the component.
<b>Block symbol</b>	Allows you to visualize the signals and interfaces added in the <b>Signals &amp; Interfaces</b> tab.
<b>Implementation Templates</b>	Allows you to export implementation templates in the form of a pre-populated HDL entity, or a template Platform Designer system which contains the boundary information (signals and interfaces) as interface requirements.
<b>Export</b>	Allows you to export the signals and interfaces of an IP component as an IP-XACT file or a <code>_hw.tcl</code> file.

**Note:** Remember to click **Apply** in the **Component Instantiation** tab for any of your changes to take effect. Alternatively, click **Revert** to undo all the changes you have made to the component.

### Related Links

[Adding a Generic Component to the Platform Designer System](#) on page 644

## 9.3.7 Connect IP Components in Your Platform Designer System

Use the **System Contents** tab to connect and configure components. Platform Designer supports connections between interfaces of compatible types and opposite directions. For example, you can connect a memory-mapped master interface to a slave interface, and an interrupt sender interface to an interrupt receiver interface. You can connect any interfaces exported from a Platform Designer system within a parent system.

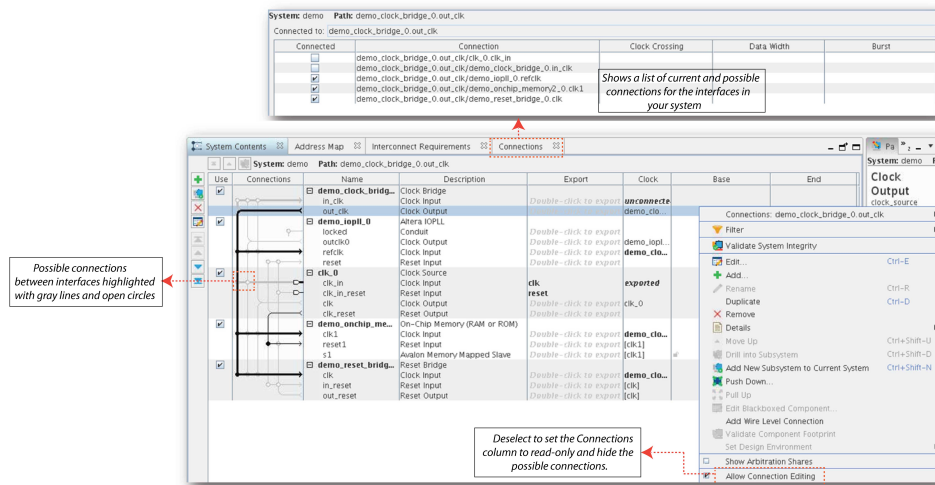
**Note:** You cannot both export and connect interfaces internally within the same Platform Designer system.

Possible connections between interfaces appear as gray lines and open circles. To make a connection, click the open circle at the intersection of the interfaces. When you make a connection, Platform Designer draws the connection line in black and fills the connection circle. Clicking a filled-in circle removes the connection.

Platform Designer takes the high-level connectivity you specify, and instantiates a suitable HDL fabric to perform the needed adaptation and arbitration between components. Platform Designer includes this interconnect fabric in the generated RTL system output. The **Connections** tab (**View > Connections**) shows a list of current and possible connections for selected instances or interfaces in the **Hierarchy** or **System Contents** tabs. You can add and remove connections by clicking the check box for each connection. Each column provides specific information about the connection. For example, the **Clock Crossing**, **Data Width**, and **Burst** columns provide interconnect information about added adapters that can result in slower  $f_{MAX}$  or increased area utilization.

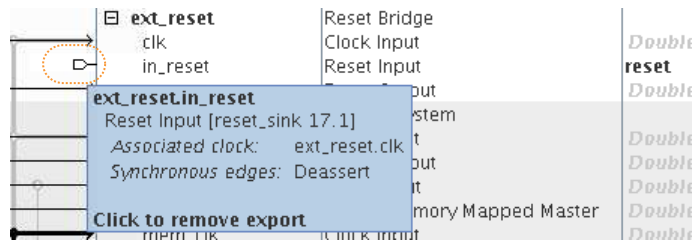
To prevent additional connectivity changes to your system, you can deselect **Allow Connection Editing** in the right-click menu. This option sets the **Connections** column to read-only and hides the possible connections.

Figure 153. Connections Column in the System Contents Tab



When you double-click an interface in the **Export** column to export it, all possible connections in the **Connections** column are displayed as a pin.

Figure 154. Connection Display for Exported Interfaces





Click the pin to restore the representation of the connections, and remove the interface from the **Export** column. You can also use the **Connections** tab to view and make connections for exported interfaces.

### 9.3.7.1 Create Connections Between Masters and Slaves

The **Address Map** tab specifies the address range that each memory-mapped master uses to connect to a slave in a Platform Designer system. Platform Designer shows the slaves on the left, the masters across the top, and the address span of the connection in each cell. If there is no connection between a master and a slave, the table cell is empty. In this case, use the **Address Map** tab to view the individual memory addresses for each connected master.

Platform Designer enables you to design a system where two masters access the same slave at different addresses. If you use this feature, Platform Designer labels the **Base** and **End** address columns in the **System Contents** tab as "mixed" rather than providing the address range.

Follow these steps to change or create a connection between master and slave IP components:

1. In Platform Designer, click the **Address Map** tab.
2. Locate the table cell that represents the connection between the master and slave component pair.
3. Either type in a base address, or update the current base address in the cell.

**Note:** The base address of a slave component must be a multiple of the address span of the component. This restriction is a requirement of the Platform Designer interconnect, which provides an efficient address decoding logic, which in turn allows Platform Designer to achieve the best possible  $f_{MAX}$ .

### 9.3.8 Validate System Integrity

The **System Messages** tab displays all the errors and warnings associated with your current Platform Designer system. Double-click the warning or error messages to open the relevant **System Contents** or **Parameters** tabs to fix the issue. You can also click validate button in the **Hierarchy** tab, or the **Validate System Integrity** button at the bottom of the main Platform Designer panel to perform system integrity check for the entire system.

**Table 93. System Messages Types in Platform Designer**

System Messages Types	Description
Component Instantiation Warning	Indicates the mismatches between system information parameters or IP core parameterization errors. A system information parameters mismatch refers to the mismatch
<i>continued...</i>	



System Messages Types	Description
	between an IP component's system parameter expectations and the component's saved system information parameters in the corresponding .ip file.
Component Instantiation Error	Indicates the mismatches between HDL entity name, compilation library, or ports which results in downstream compilation errors. The component instantiation errors always indicate the fundamental mismatches between generated system and interconnect fabric RTL.
System Connectivity Warning	Platform Designer system connectivity warnings.
System Connectivity Error	Platform Designer system connectivity errors.

### 9.3.8.1 Component Instantiation Warning Messages

Component Instantiation Warnings report the following inconsistencies:

- Interface types do not match
- Interface is missing
- Port has been moved to another interface
- Port role has changed
- Interface assignment is mismatched
- Interface assignment is missing

### 9.3.8.2 Component Instantiation Error Messages

Component Instantiation Errors report the following inconsistencies:

- Port is missing from the ip file
- Port is missing from instantiation
- Port direction has changed
- Port VHDL type has changed
- Port width has changed
- Interface Parameter is mismatched
- Interface Parameter is missing

### 9.3.8.3 Validate System Integrity for Individual Components in the System

To validate the system integrity for your IP components:

1. Select the IP component in the **System Contents** tab.
2. Right-click and select **Validate Component Footprint** to check for any mismatches between the IP component and its .ip file representation.
3. If there are any errors, click **Reload Component Footprint** to reload the signals and interfaces for the component from the .ip file.

*Note:* To perform system integrity check for the entire system, right-click the **System Contents** tab and select **Validate System Integrity**. You can also click the validate button in the **Hierarchy** tab, or the **Validate System Integrity** button at the bottom of the main Platform Designer panel.

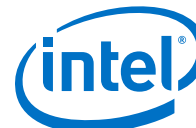
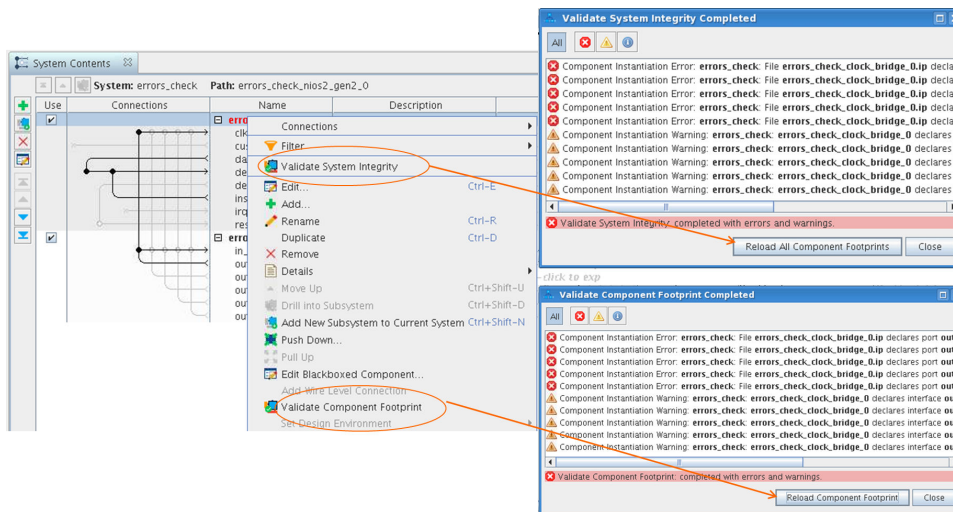


Figure 155. Validating System Integrity



### 9.3.9 Propagate System Information to IP Components

When system information doesn't match the requirements of an IP component, use the **System Info** tab to synchronize the IP component with mismatches. To open the **System Info** tab, click **View > System Info**.

Table 94. System Info GUI Information

Name	Description
<b>Component Instantiation</b>	This table shows the signals and interfaces for the selected IP component within the system. Mismatches are highlighted in blue. Missing elements are highlighted in green.
<b>IP file</b>	This table shows the signal and interface information for the selected IP component from its corresponding .ip file. Mismatches are highlighted in blue. Missing elements are highlighted in green.
<b>Component Instantiation Value</b>	This table shows the selected interface parameter value of the IP component within the system.
<b>IP File Value</b>	This table shows the selected interface parameter value of the IP component from the corresponding .ip file.
>>	This button allows you to manually synchronize the mismatches in signals and interfaces, one at a time, between the IP file and the IP component.
<b>Sync All</b>	This button allows you to synchronize all the system info mismatches for the IP component.

**Note:** To update the system information for all the IP components in your current system simultaneously, click the update icon in the **Hierarchy** tab or the **Sync All System Info** button at the bottom of the main Platform Designer panel.

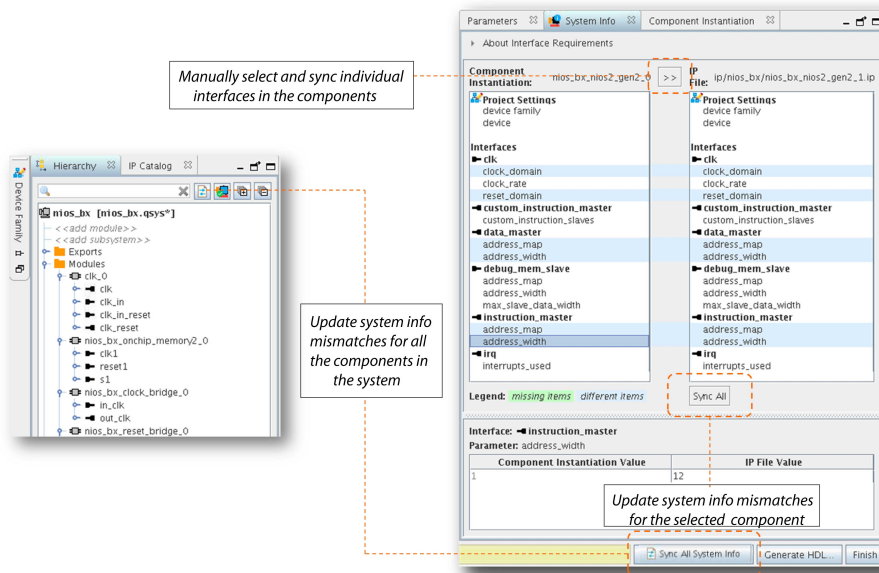
### 9.3.9.1 Update System Information

If the system information does not match the saved requirements of the corresponding .ip file for an IP component, the mismatches appear as Component Instantiation Warnings in the **System Messages** tab. In Platform Designer, you must manually synchronize these system info dependencies:

1. To open the **System Info** tab, select the signal or interface in the **System Contents** tab and click **View > System Info**. You can also double-click the corresponding Component Instantiation Warning in the **System Messages** tab to open the system-info mismatch information in the **System Info** tab.
2. To update the .ip file with the current system information, select the mismatched parameter and click >>. Alternatively, you can synchronize all the mismatches for the component by clicking the **Sync All** button.
3. To update the system information for all the IP components in your current system, click **Sync All System Info** in the bottom right corner of the Platform Designer main frame.

*Note:* Clicking the update icon near the search field in the **Hierarchy** tab also synchronizes the system information for all the IP components in your system.

Figure 156. Updating System Information



### 9.3.10 View Your Platform Designer System

Platform Designer allows you to change the display of your system to match your design development. Each tab on **View** menu allows you to view your design with a unique perspective. Multiple tabs open in your workspace allows you to focus on a selected element in your system under different perspectives.





The Platform Designer GUI supports global selection and edit. When you make a selection or apply an edit in the **Hierarchy** tab, Platform Designer updates all other open tabs to reflect your action. For example, when you select `cpu_0` in the **Hierarchy** tab, Platform Designer updates the **Parameters** tab to show the parameters for `cpu_0`.

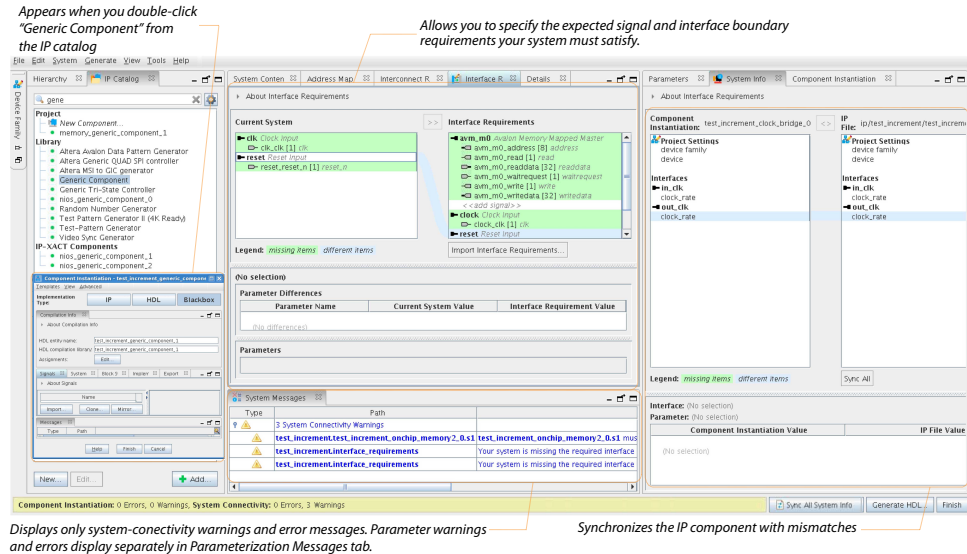
- By default, when you open Platform Designer, the **IP Catalog**, **Hierarchy**, and the **Device Family** tabs appear to the left of the main frame.
- The **System Contents**, **Address Map**, **Interconnect Requirements**, and **Details** tabs display in the main frame.
- **Parameters**, **System Info**, and **Component Instantiation** tabs appear to the right of the main frame.
- The **System Messages** tab displays in the lower portion of Platform Designer.
- The **Parameterization Messages** tab appears in the lower portion of the **Parameter** tab when you select an IP component, displaying parameter warnings and error messages, specific to that component.

*Note:* The **Parameterization Messages** tab also appears in the bottom pane of the parameter editor when you double-click an IP component from the IP Catalog.

You can dock tabs in the main frame as a group, or individually by clicking the tab control in the upper-right corner of the main frame. You can arrange your workspace by dragging and dropping, and then grouping tabs in an order appropriate to your design development, or close or dock tabs that you are not using. Tool tips on the upper-right corner of the tab describe possible workspace arrangements, for example, restoring or disconnecting a tab to or from your workspace. When you save your system, Platform Designer also saves the current workspace configuration. When you re-open a saved system, Platform Designer restores the last saved workspace.

The **Reset to System Layout** command on the **View** menu restores the workspace to its default configuration for Platform Designer system design. The **Reset to IP Layout** command restores the workspace to its default configuration for defining and generating single IP cores.

Figure 157. Platform Designer GUI



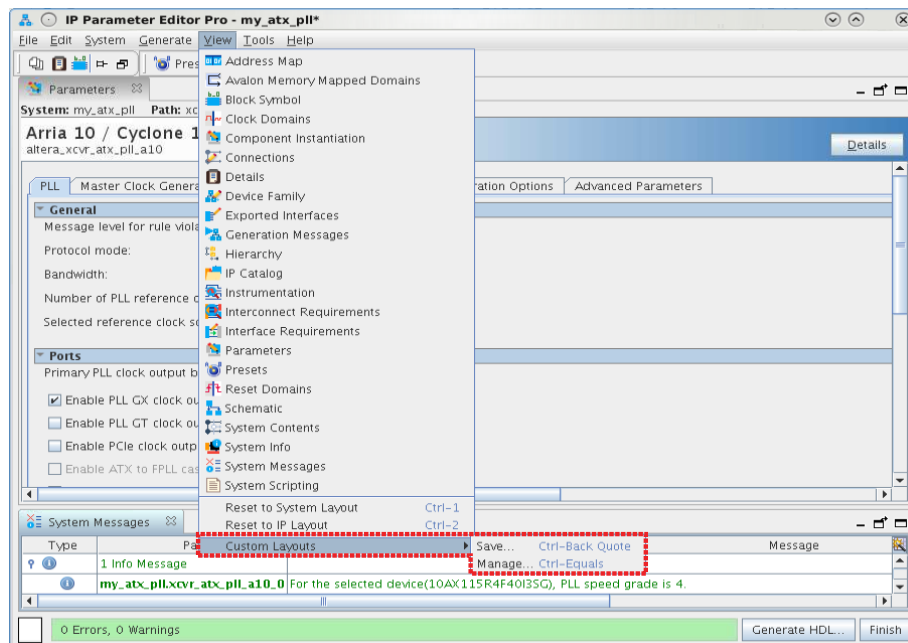
### 9.3.10.1 Manage Platform Designer Window Views with Layouts

Platform Designer Layout controls what tabs are open in your Platform Designer design window. When you create a Platform Designer window configuration that you want to keep, Platform Designer allows you to save that configuration as a custom layout. The Platform Designer GUI and features are well-suited for Platform Designer system design. You can also use Platform Designer to define and generate single IP cores for use in your Intel Quartus Prime software projects.

1. To configure your Platform Designer window with a layout suitable for Platform Designer system design, click **View > Reset to System Layout**. The **System Contents**, **Address Map**, **Interconnect Requirements**, and **Messages** tabs open in the main pane, and the **IP Catalog** and **Hierarchy** tabs along the left pane.
2. To configure your Platform Designer window with a layout suitable for single IP core design, click **View > Reset to IP Layout**. The **Parameters** and **Messages** tabs open in the main pane, and the **Details**, **Block Symbol** and **Presets** tabs along the right pane.
3. To save your current Platform Designer window configuration as a custom layout, click **View > Custom Layouts > Save**. Platform Designer saves your custom layout in your project directory, and adds the layout to the custom layouts list, and the `layouts.ini` file. The `layouts.ini` file controls the order in which the layouts appear in the list.
4. To reset your Platform Designer window configuration to a previously saved configuration, click **View > Custom Layouts**, and then select the custom layout in the list. The Platform Designer windows opens with your previously saved Platform Designer window configuration.



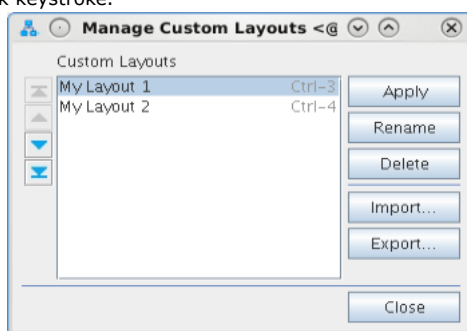
Figure 158. Save Your Platform Designer Window Views and Layouts



- To manage your saved custom layouts, click **View > Custom Layouts**. The **Manage Custom Layouts** dialog box opens and allows you to apply a variety of functions that facilitate custom layout management. For example, you can import or export a layout from or to a different directory.

Figure 159. Manage Custom Layouts

The shortcut, **Ctrl-3**, for example, allows you to quickly change your Platform Designer window view with a quick keystroke.

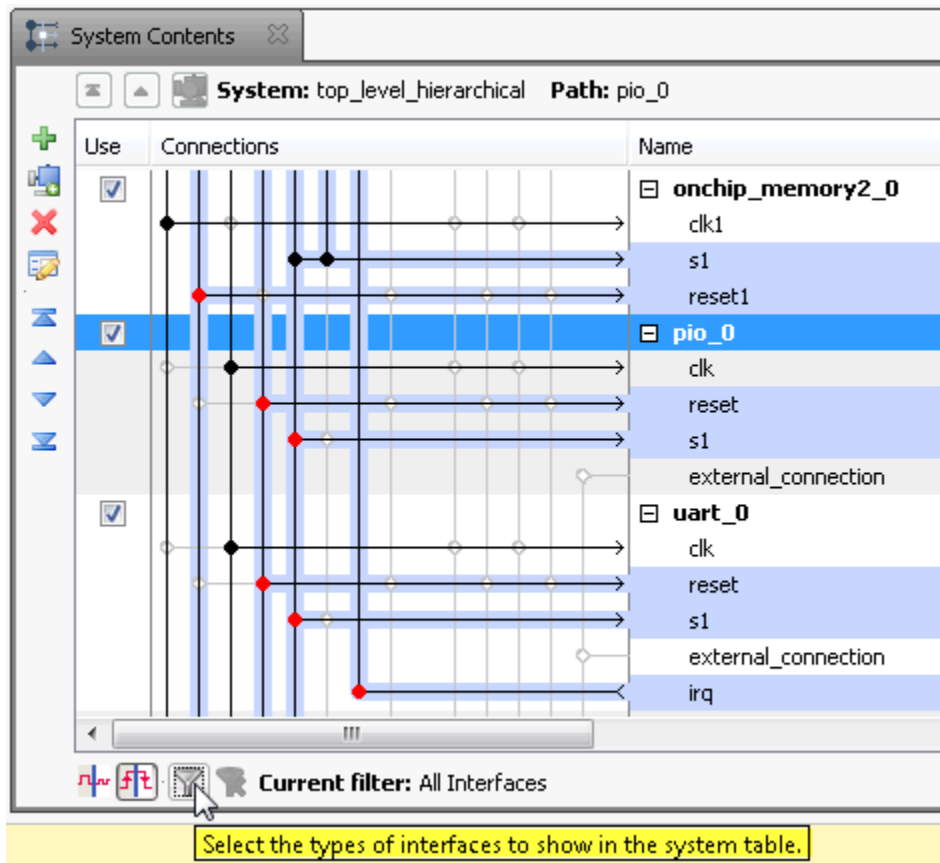


### 9.3.10.2 Filter the Display of the System Contents Tab

You can use the **Filters** dialog box to filter the display of your system by interface type, instance name, or by using custom tags.

For example, in the **System Contents** tab, you can show only instances that include memory-mapped interfaces or instances connected to a particular Nios II processor. The filter tool also allows you to temporarily hide clock and reset interfaces to simplify the display.

Figure 160. Filter Icon in the System Contents Tab



### Related Links

[Filters Dialog Box](#)

### 9.3.10.3 Display Details About a Component or Parameter

The **Details** tab provides information for a selected component or parameter. Platform Designer updates the information in the **Details** tab as you select different components.

As you click through the parameters for a component in the parameter editor, Platform Designer displays the description of the parameter in the **Details** tab. To return to the complete description for the component, click the header in the **Parameters** tab.

### 9.3.10.4 Display a Graphical Representation of a Component

In the **Block Symbol** tab, Platform Designer displays a graphical representation of the element that you select in the **Hierarchy** or **System Contents** tabs. You can view the selected component's port interfaces and signals. The **Show signals** option allows you to turn on or off signal graphics.

The **Block Symbol** tab appears by default in the parameter editor when you add a component to your system. When the **Block Symbol** tab is open in your workspace, it reflects changes that you make in other tabs.

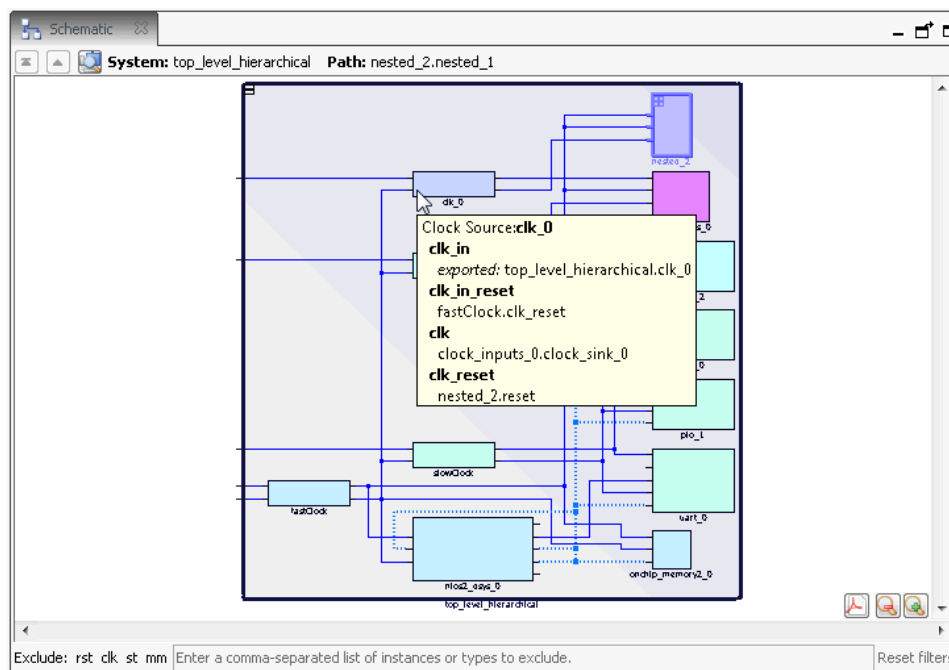


### 9.3.10.5 View a Schematic of Your Platform Designer System

The **Schematic** tab displays a schematic representation of your Platform Designer system. Tab controls allow you to zoom into a component or connection, or to obtain tooltip details for your selection. You can use the image handles in the right panel to resize the schematic image.

If your selection is a subsystem, use the Hierarchy tool to navigate to the parent subsystem, move up one level, or to drill into the currently open subsystem.

Figure 161. Platform Designer Schematic Tab



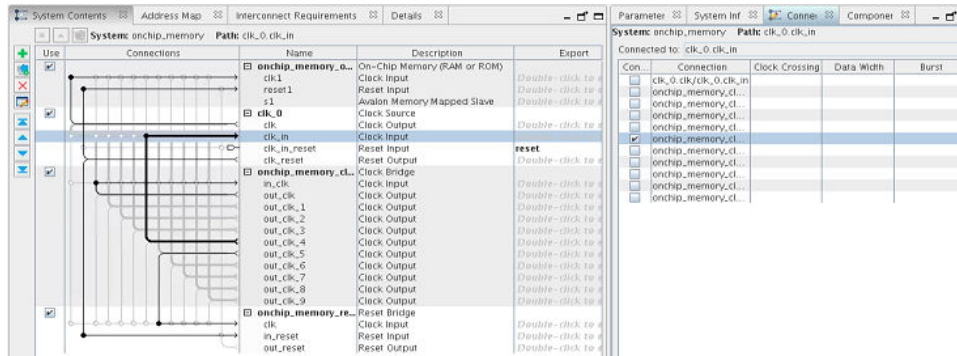
#### Related Links

[Edit a Platform Designer Subsystem on page 360](#)

### 9.3.10.6 View Connections in Your Platform Designer System

The **Connections** tab displays a lists of connections in your Platform Designer system. On the **Connections** tab (**View > Connections**), you can choose to connect or un-connect a module in your system, and then view the results in the **System Contents** tab.

Figure 162. Connections tabs in Platform Designer



### 9.3.11 Navigate Your Platform Designer System

The **Hierarchy** tab is a full system hierarchical navigator that expands the Platform Designer system contents to show all elements in your system.

You can use the **Hierarchy** tab to browse, connect, parameterize IP, and drive changes in other open tabs. Expanding each interface in the **Hierarchy** tab allows you to view sub-components, associated elements, and signals for the interface. You can focus on a particular area of your system by coordinating selections in the **Hierarchy** tab with other open tabs in your workspace.

Navigating your system using the **Hierarchy** tab in conjunction with relevant tabs is useful during the debugging phase. Viewing your system with multiple tabs open allows you to focus your debugging efforts to a single element in your system.

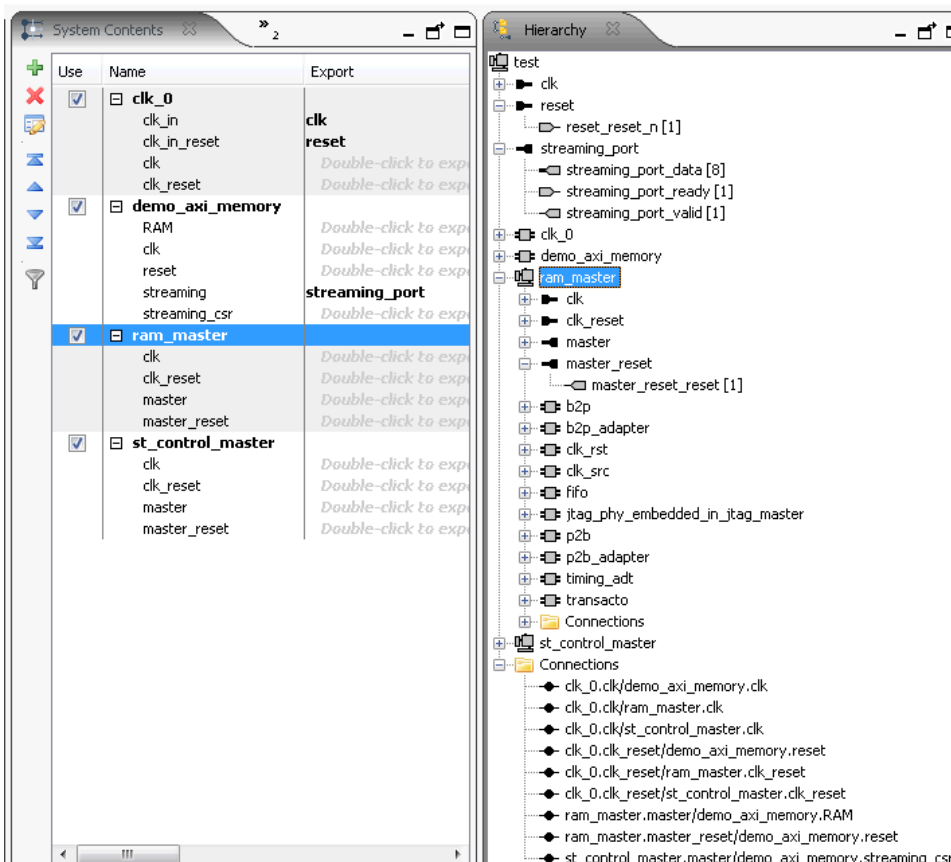
The **Hierarchy** tab provides the following information and functionality:

- Connections between signals.
- Names of signals in exported interfaces.
- Right-click menu to connect, edit, add, remove, or duplicate elements in the hierarchy.
- Internal connections of Platform Designer subsystems that are included as IP components. In contrast, the **System Contents** tab displays only the exported interfaces of Platform Designer subsystems.



**Figure 163. Expanding System Contents in the Hierarchy Tab**

The **Hierarchy** tab displays a unique icon for each element in the system. Context sensitivity between tabs facilitates design development and debugging. For example, when you select an element in the **Hierarchy** tab, Platform Designer selects the same element in other open tabs. This allows you to interact with your system in more detail. In the example below, the `ram_master` selection appears selected in both the **System Contents** and **Hierarchy** tabs.



**Related Links**

[Create and Manage Hierarchical Platform Designer Systems on page 358](#)

**9.3.12 Specify IP Component Parameters**

The **Parameters** tab allows you to configure parameters that define an IP component's functionality.

When you add a component to your system, or when you double-click a component in an open tab, the parameter editor opens. In the parameter editor, you can configure the parameters of the component to align with the requirements of your design. If you create your own IP components, use the Hardware Component Description File (`_hw.tcl`) to specify configurable parameters.

Whenever you add an IP component to your system, Platform Designer stores the instantiated IP component in a separate `.ip` file. Any changes you make to the component's parameters from the **Parameters** tab, automatically updates the corresponding `.ip` file.

With the **Parameters** tab open, when you select an element in the **Hierarchy** tab, Platform Designer shows the same element in the **Parameters** tab. You can then make changes to the parameters that appear in the parameter editor, including changing the name for top-level instance that appears in the **System Contents** tab. Changes that you make in the **Parameters** tab affect your entire system and appear dynamically in other open tabs in your workspace.

In the parameter editor, the **Documentation** button provides information about a component's parameters, including the version.

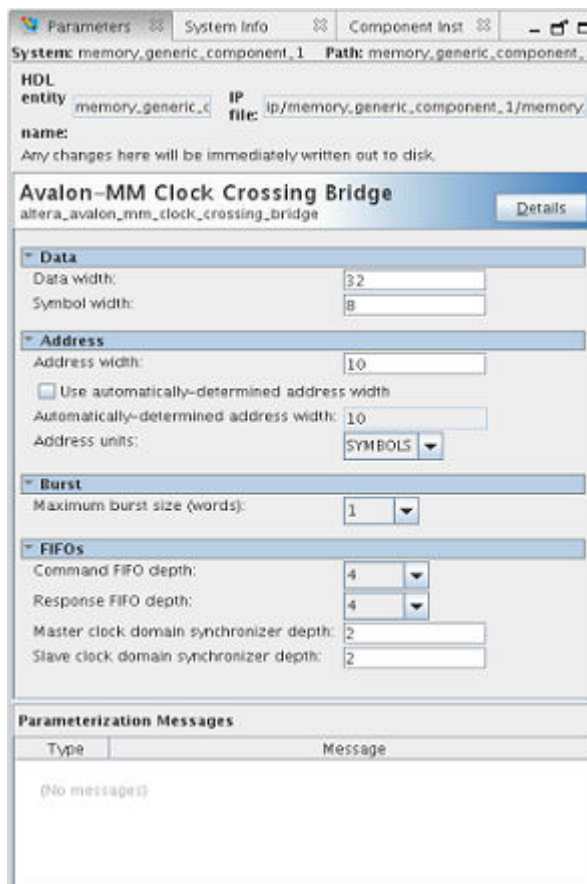
At the top of the parameter editor, Platform Designer shows the hierarchical path for the component and its elements. This feature is useful when you navigate deep within your system with the **Hierarchy** tab.

Below the hierarchical path, the parameter editor shows the HDL entity name and the IP file path for the selected IP component.

The **Parameters** tab also allows you to review the timing for an interface and displays the read and write waveforms at the bottom of the **Parameters** tab.

The **Parameterization Messages** appears at lower portion of the parameter editor, displaying parameter warnings and error messages, specific to the selected IP component.

**Figure 164. Avalon-MM Write Master Timing Waveforms in the Parameters Tab**







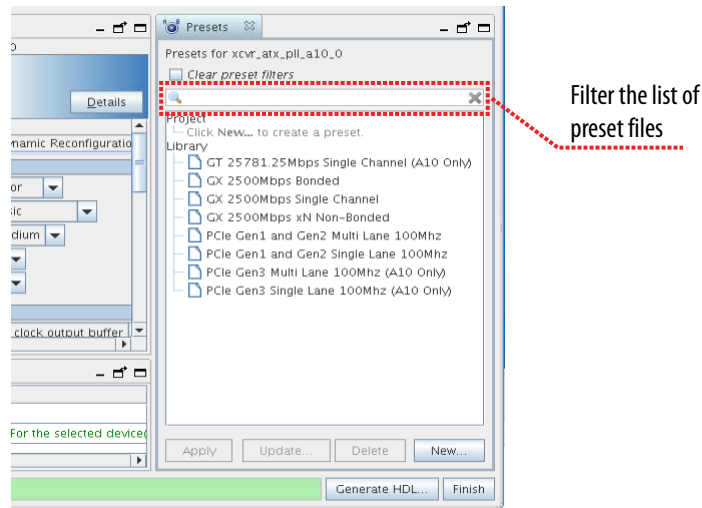
### 9.3.12.1 Configure Your IP Component with a Pre-Defined Set of Parameters

The **Presets** tab allows you to apply a pre-defined set of parameters to your IP component to create a unique variation. The **Presets** tab opens the preset editor and allows you to create, modify, and save custom component parameter values as a preset file. Not all IP components have preset files.

When you add a new component to your system, if there are preset files available for the component, the preset editor opens in the parameter editor. The name of each preset file describes a particular protocol.

1. In your Platform Designer system, select an element in the **Hierarchy** tab.
2. Click **View > Presets**.
3. Type text in the **Presets** search box to filter the list of preset files.  
For example, if you add the **DDR3 SDRAM Controller with UniPHY** component to your system, type `1g micron 256` in the search box, The **Presets** list displays only those preset files associated with `1g micron 256`.
4. Click **Apply** to assign the selected presets to the component.  
Presets whose parameter values match the current parameter settings appear in bold.
5. In the **Presets** tab, click **New** to create a custom preset file if the available presets do not meet the requirements of your design.
  - a. In the **New Preset** dialog box, specify the **Preset name** and **Preset description**.
  - b. Check or uncheck the parameters you want to include in the preset file.
  - c. Specify where you want to save the new preset file.  
If the file location that you specify is not already in the IP search path, Platform Designer adds the location of the new preset file to the IP search path.
  - d. Click **Save**.
6. In the **Presets** tab, click **Update** to update a custom preset.  
*Note:* Custom presets are preset files that you create by clicking **New** in the **Presets** tab.
7. In the **Presets** tab, click **Delete** to delete a custom preset.

Figure 165. Specifying Presets



### 9.3.13 Modify an Instantiated IP Component

Platform Designer allows you to manipulate the system representation of IP components. For example, you can modify the interfaces of an instantiated IP component to change its properties.

The example below shows how to instantiate a PLL in your system and then modify its conduit interface so that the conduit becomes a reset.

#### 9.3.13.1 Change a Conduit to a Reset

1. In the IP Catalog search box, locate **Altera IOPLL** and double-click to add the component to your system.
2. Select the **PLL** component in the **System Contents** tab.
3. Open the **Component Instantiation** tab for the selected component.  
*Note:* The **Component Instantiation** tab displays in the right pane of the Platform Designer window. If you can't find the tab on the main frame of Platform Designer, click **View > Component Instantiation** to open the tab.
4. In the **Signals & Interfaces** tab, select the **locked** conduit interface.
5. Change the **Type** from **Conduit** to **Reset Input**, and the **Synchronous edges** from **Deassert** to **None**.
6. Select the locked [1] signal below the **locked** interface.
7. Change the **Signal Type** from **export** to **reset\_n**. Change the **Direction** from **output** to **input**.
8. Click **Apply**.

The conduit interface changes to reset for the instantiated PLL component.

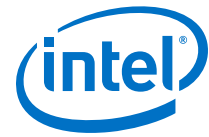
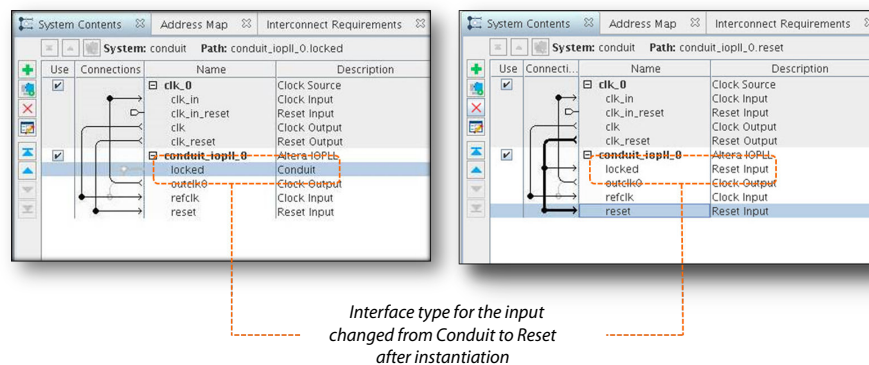


Figure 166. Changing Conduit to a Reset



### 9.3.14 Save your System

To save your Platform Designer system, click **File > Save**. To save a standalone .ip file that you open in the IP Parameter Editor Pro window, click **File > Save**. To create a copy of the standalone .ip file, click **File > Save As**.

Note:

- To save a copy of the Platform Designer system, refer to the *Archive your System* section.
- To save the system as a Platform Designer script, click **File > Export System as qsys script (.tcl)**. You can restore this system by executing the .tcl script from the **System Scripting** tab.

#### Related Links

[Archive your System](#) on page 355

### 9.3.15 Archive your System

Platform Designer allows you to archive your system in a .zip format. To archive your system, click **File > Archive System**.

In the **Archive System** dialog box, the **Collect to common directory** option is turned on by default. This option allows Platform Designer to collect all the .qsys files in the root directory of the archive, and all the .ip files to a single ip directory, while updating all the references to match. Disable this option to maintain the current directory structure for the archive.

To extract all the archived files in a given system to a specified folder, click **File > Restore Archive System**. Select the source archive file, and the destination folder. Upon successful extraction, Platform Designer automatically launches the **Open System** dialog box, with the extracted .qsys file and the associated .qpf file, preloaded.

Note:

You can also archive your system using command-line options. For more information, refer to *Archive a System with qsys-archive* section.

#### Related Links

[Archive a Platform Designer System with qsys-archive](#) on page 602

## 9.4 Synchronize IP File References

Whenever you load a system, Platform Designer ensures that the referenced IP files in your Platform Designer system matches the IP files list in the associated Intel Quartus Prime project.

The **IP Synchronization Result** dialog box displays the discrepancies list whenever IP synchronization mismatches occur in your Platform Designer system. To manually check for these mismatches, click **File > Synchronize IP File References**.

Platform Designer identifies the following types of mismatches with the IP synchronization:

**Table 95. IP Synchronization Results**

Mismatch Type	Description
<b>Duplicate IP files</b>	The list of same IP files references specified in both your Platform Designer system and the associated Intel Quartus Prime project. These IP files contain the same name, but are present in different locations. In such cases, the IP files referenced in the Intel Quartus Prime project takes precedence. Platform Designer replaces the IP file reference in the system with the one in the Intel Quartus Prime project. <i>Note:</i> If the Intel Quartus Prime project contains more than one IP of the same file name, Platform Designer retains the first instance and removes all other occurrences of the IP file with the specific name.
<b>Missing IP files</b>	The list of missing IP file references specified in both your Platform Designer system and the corresponding Intel Quartus Prime project. In such cases, Platform Designer allows you to select a replacement IP file.
<b>Missing Platform Designer IP files</b>	The list of missing IP file references in your Platform Designer system whose associated Intel Quartus Prime project contains valid IP files of the same names. If Platform Designer locates a valid reference in the Intel Quartus Prime project, it replaces the missing reference in the Platform Designer system with IP file reference from the Intel Quartus Prime project.
<b>Missing Quartus IP files</b>	The list of IP file references in your Platform Designer system which are not listed in the associated Intel Quartus Prime project's .qsf file. Platform Designer adds the missing IP file reference to the Intel Quartus Prime project. If the project's .qsf file already contains reference to the missing IP file, but the file cannot be located in the specified path, Platform Designer removes the reference in the .qsf file, and adds the reference to the IP file in the Platform Designer system.

## 9.5 Upgrade Outdated IP Components in Platform Designer

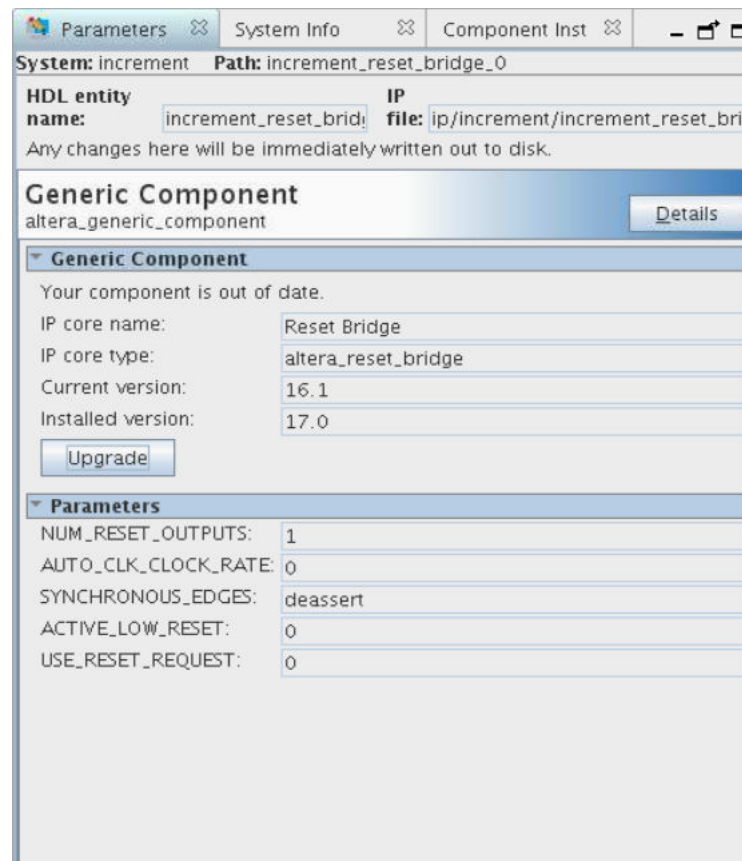
When you open a Platform Designer system containing outdated IP components, you can retain and use the RTL of previously generated IP components within the Platform Designer system. If Platform Designer is unable to locate the IP core's original version, you cannot re-parametrize the IP core without upgrading the IP core to the latest version. However, Platform Designer allows you to view the parametrization of the original core without upgrading.



To upgrade individual IP components in your Platform Designer system:

1. Click **View** > **Parameters**
2. Select the outdated IP component in the **Hierarchy** or the **System Contents** tab.
3. Click the **Parameters** tab. This tab displays information on the current version, as well as the installed version of the selected IP component.
4. Click **Upgrade**. Platform Designer upgrades the IP component to the installed version, and deletes all the RTL files associated with the IP component.

**Figure 167. Upgrade IP Component in your Platform Designer System**



To upgrade an IP component from the command-line, type the following:

```
qsys-generate --upgrade-ip-cores <ip_file>
```

To upgrade all the IP components in your Platform Designer system, open the associated project in the Intel Quartus Prime software, and click **Project** > **Upgrade IP Components**.

### Related Links

[Introduction to the Platform Designer IP Catalog](#) on page 329

## 9.6 Create and Manage Hierarchical Platform Designer Systems

Platform Designer supports hierarchical system design. You can add any Platform Designer system as a subsystem in another Platform Designer system. Platform Designer hierarchical system design allows you to create, explore and edit hierarchies dynamically within a single instance of the Platform Designer editor. Platform Designer generates the complete hierarchy during the top-level system's generation.

**Note:** You can explore parameterizable Platform Designer systems and `_hw.tcl` files, but you cannot edit their elements.

Your Platform Designer systems appear in the IP Catalog under the System category under Project. You can reuse systems across multiple designs. In a team-based hierarchical design flow, you can divide large designs into subsystems and have team members develop subsystems simultaneously.

### Related Links

[Navigate Your Platform Designer System](#) on page 350

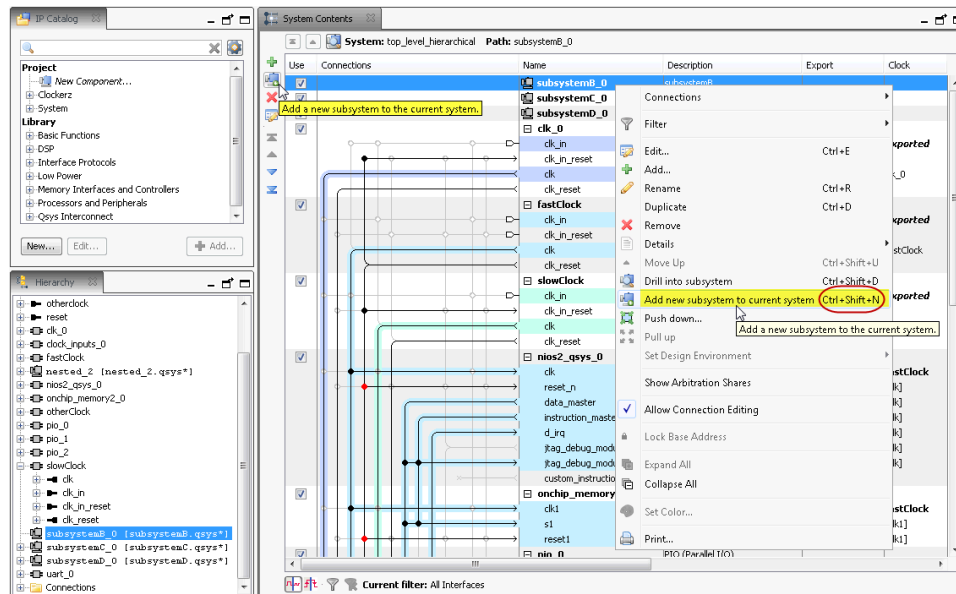
### 9.6.1 Add a Subsystem to Your Platform Designer Design

You can create a child subsystem or nest subsystems at any level in the hierarchy. Platform Designer adds a subsystem to the system you are currently editing. This can be the top-level system, or a subsystem.

To create or nest subsystems in your Platform Designer design, use the following methods within the **System Contents** tab:

- Right-click command: **Add a new subsystem to the current system.**
- Left panel icon.
- **CTRL+SHIFT+N.**

**Figure 168. Add a Subsystem to Your Platform Designer Design**





## 9.6.2 Drill into a Platform Designer Subsystem to Explore its Contents

The ability to drill into a system provides visibility into its elements and connections. When you drill into an instance, you open the system it instantiates for editing.

You can drill into a subsystem with the following commands:

- Double-click a system in the **Hierarchy** tab.
- Right-click a system in the **System Contents** or **Schematic** tabs, and then select **Drill into subsystem**.
- CTRL+SHIFT+D in the **System Contents** tab.

*Note:* You can only drill into `.qsys` files, not parameterizable Platform Designer systems or `_hw.tcl` files.

The **Hierarchy** tab is rooted at the top-level and drives global selection. You can manage a hierarchical Platform Designer system that you build across multiple Platform Designer files, and view and edit their interconnected paths and address maps simultaneously. As an example, you can select a path to a subsystem in the **Hierarchy** tab, and then drill deeper into the subsystem in the **System Contents** or **Schematic** tabs.

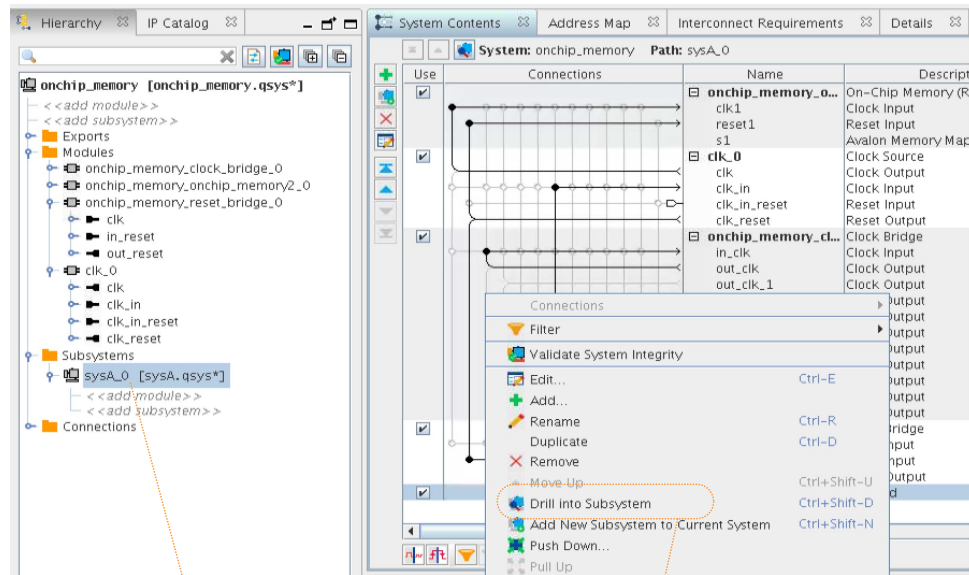
Views that manage system-level editing, for example, the **System Contents** and **Schematic** tabs, contain the hierarchy widget, which allows you to efficiently navigate your subsystems. The hierarchy widget also displays the name of the current selection, and its path in the context of the system or subsystem.

The widget contains the following controls and information:

- **Top**—Navigates to the project-level `.qsys` file that contains the subsystem.
- **Up**—Navigates up one level from the current selection.
- **Drill Into**—Allows you to drill into an editable system.
- **System**—Displays the hierarchical location of the system you are currently editing.
- **Path**—Displays the relative path to the current selection.

*Note:* In the **System Contents** tab, you can use CTRL+SHIFT+U to navigate up one level, and CTRL+SHIFT+D to drill into a system.

Figure 169. Drill into a Platform Designer System to Explore its Contents



Double-click the Subsystem from the Hierarchy Tab or  
 Select Drill into Subsystem Option by Right-Clicking  
 System Contents Tab to Switch Subsystems

### 9.6.3 Edit a Platform Designer Subsystem

You can double-click a Platform Designer subsystem in the **Hierarchy** tab to edit its contents in any tab. When you make a change, open tabs refresh their content to reflect your edit. You can change the level of a subsystem, or push it into another subsystem with commands in the **System Contents** tab.

**Note:**

To edit a .qsys file, the file must be writeable and reside outside of the ACDS installation directory. You cannot edit systems that you create from composed \_hw.tcl files, or systems that define instance parameters.



1. In the **System Contents** or **Schematic** tabs, use the hierarchy widget to navigate to the top-level system, up one level, or down one level (drill into a system). All tabs refresh and display the requested hierarchy level.
2. To edit a system, double-click the system in the **Hierarchy** tab. You can also drill into the system with the Hierarchy tool or right-click commands, which are available in the **Hierarchy**, **Schematic**, **System Contents** tabs. The system is open and available for edit in all Platform Designer views. A system currently open for edit appears as bold in the **Hierarchy** tab.
3. In the **System Contents** tab, you can rename any element, add, remove, or duplicate connections, and export interfaces, as appropriate. Changes to a subsystem affect all instances. Platform Designer identifies unsaved changes to a subsystem with an asterisk next to the subsystem in the **Hierarchy** tab.





### Related Links

[View a Schematic of Your Platform Designer System](#) on page 349

## 9.6.4 Change the Hierarchy Level of a Platform Designer Component

You can push selected components down into their own subsystem, which can simplify your top-level system view. Similarly, you can pull a component up out of a subsystem to perhaps share it between two unique subsystems. Hierarchical-level management facilitates system optimization and can reduce complex connectivity in your subsystems. When you make a change, open tabs refresh their content to reflect your edit.

1. In the **System Contents** tab, to group multiple components that perhaps share a system-level component, select the components, right-click, and then select **Push down into new subsystem**.  
Platform Designer pushes the components into their own subsystem and re-establishes the exported signals and connectivity in the new location.
2. In the **System Contents** tab, to pull a component up out of a subsystem, select the component, and then click **Pull up**.  
Platform Designer pulls the component up out of the subsystem and re-establishes the exported signals and connectivity in the new location.

## 9.6.5 Save New Platform Designer Subsystem

When you save a subsystem to your Platform Designer design, Platform Designer confirms the new subsystem in the **Confirm New System Filenames** dialog box. The **Confirm New System Filenames** dialog box appears when you save your Platform Designer design. Platform Designer uses the name that you give a subsystem as `.qsys` filename, and saves the subsystem in the project's `ip` directory.

1. Click **File > Save** to save your Platform Designer design.
2. In the **Confirm New System Filenames** dialog box, click **OK** to accept the subsystem file names.  
*Note:* If you have not yet saved your top-level system, or multiple subsystems, you can type a name, and then press **Enter**, to move to the next un-named system.
3. In the **Confirm New System Filenames** dialog box, to edit the name of a subsystem, click the subsystem, and then type the new name.
4. To cancel the save process, click **Cancel** in the **Confirm New System Filenames** dialog box.

## 9.7 Specify Signal and Interface Boundary Requirements

The **Interface Requirements** tab allows you to specify the expected signal and interface boundary requirements that your Platform Designer system must satisfy. Use this tab to view and resolve any interface requirement mismatches in your current system. You can also edit the names of the exported signals and interfaces in your system from the **Interface Requirements** tab.

To open the **Interface Requirements** tab, click **View > Interface Requirements**.

**Table 96. Interface Requirements GUI Information**

Name	Description
<b>Current System</b>	This table displays all the exported interfaces in your current Platform Designer system. Add or remove the interfaces in the <b>Current System</b> table by adding or removing instances to the system in the <b>System Contents</b> tab.
<b>Interface Requirements</b>	This table shows all the interface requirements set for the current Platform Designer system.
<b>Parameter Differences</b>	This table lists the <b>Parameter Name</b> , <b>Current System Value</b> , and <b>Interface Requirement Value</b> for the selected mismatched interface. <i>Note:</i> The <b>Interface Requirements</b> tab highlights in blue the signals and interfaces that are the same, but have different parameter values. Selecting a blue item populates the <b>Parameter Differences</b> table.
<b>Import Interface Requirements</b>	This button allows you to populate the <b>Interface Requirements</b> table from an IP-XACT file representing a generic component or an entire Platform Designer system.
<b>Parameters</b>	This table lists the signal and interface parameters for the selected interface. You can view the table as <b>Current Parameters</b> when you select an interface or signal from the <b>Current System</b> table, and as <b>Required Parameters</b> when you select the signal or interface from <b>Interface Requirements</b> table. You can modify the name of your exported signal or interface from this table. For more information about how to edit the name of an exported signal or interface, refer to <i>Editing the Name of Exported Interfaces and Signals</i> in volume 1 of the <i>Intel Quartus Prime Pro Edition Handbook</i> .

### 9.7.1 Match the Exported Interface with Interface Requirements

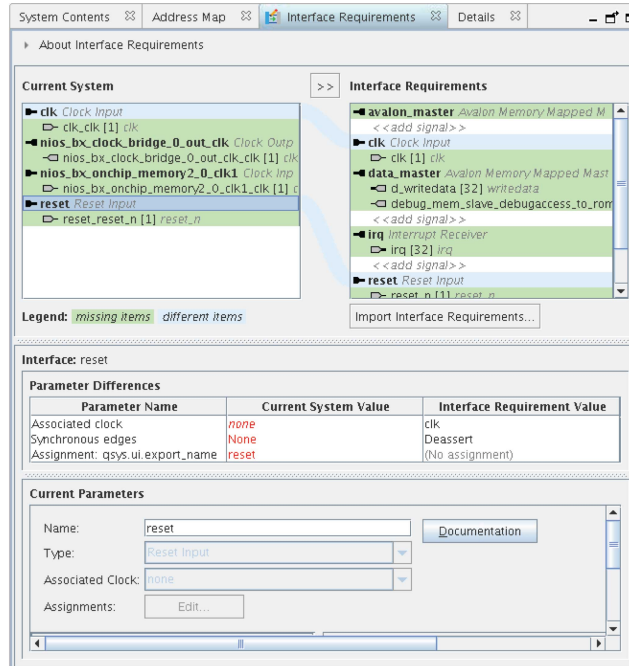
If an exported interface does not match the interface requirements of the system, Platform Designer generates component instantiation errors. You must match all the exported interfaces with the interface requirements of the system:

1. To open the **Interface Requirements** tab, click **View > Interface Requirements**.
2. To load the interface requirements from a Platform Designer system, click **Import Interface Requirements** in the **Interface Requirements** table. A dialog box appears from which you can choose the .ipxact representation of the Platform Designer system.
3. To add new interface requirements, click <<add interface>> or <<add signal>> in the **Interface Requirements** table.
4. To correct the mismatches, select the missing or mismatched interface or signal in the **Current System** table and click >>.

*Note:* Platform Designer highlights the mismatches between the system and interface requirements in blue, and highlights the missing interfaces and signals in green.



Figure 170. Interface Requirements Tab



### Related Links

- [Specify Signal and Interface Boundary Requirements](#) on page 361
- [Creating System Template for a Generic Component](#) on page 655

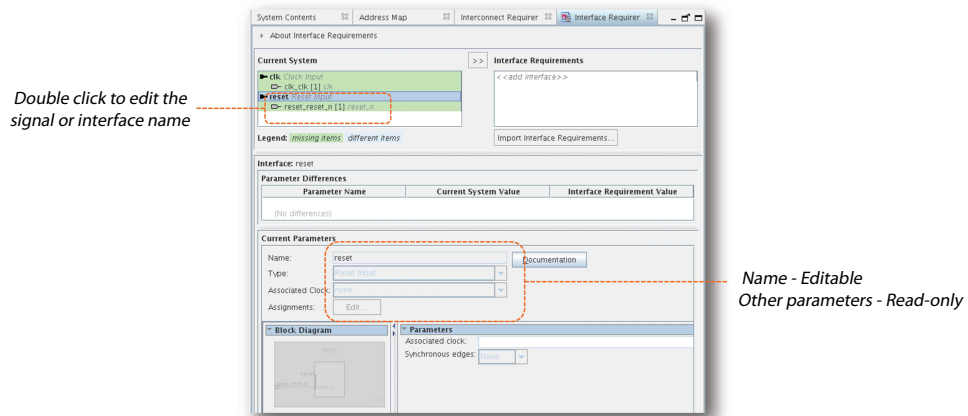
## 9.7.2 Edit the Name of Exported Interfaces and Signals

To rename the exported signal or interface:

- Double-click the signal or interface in **Current System** table.
- Select the signal or interface in the **Current System** table and press F2.
- Select the signal or interface in the **Current System** table and rename from the **Current Parameters** pane at the bottom of the tab. The **Current Parameters** pane displays all the parameters of the selected interface or signal.

*Note:* All other parameters in the **Current Parameters** except **Name** are read-only for the current system.

Figure 171. Editing the Name of Exported Interfaces and Signals



## 9.8 Run System Scripts

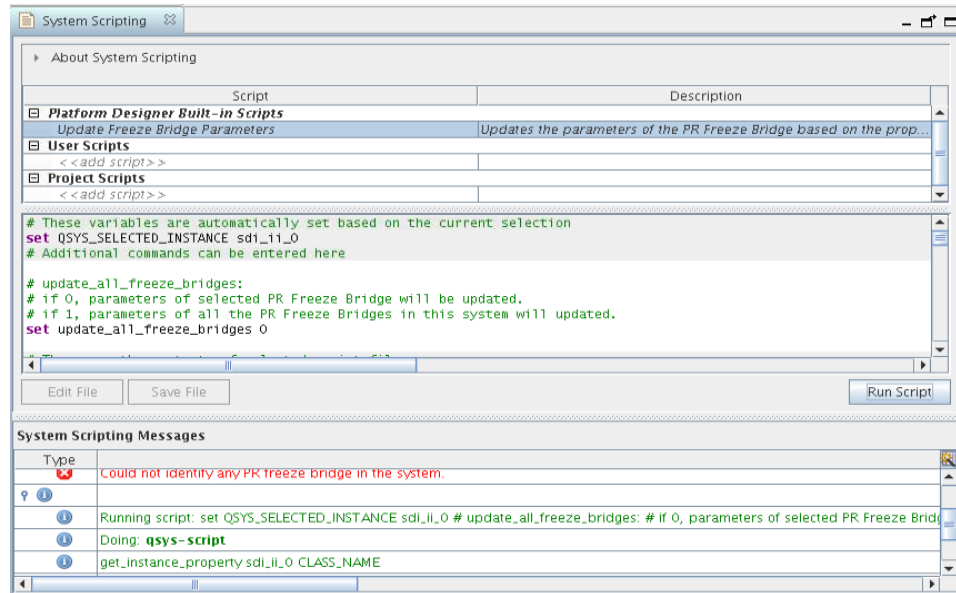
The **System Scripting** tab allows you to execute Tcl scripts on your Platform Designer system. To open the **System Scripting** tab, click **View > System Scripting**.

Table 97. System Scripting GUI Information

Name	Description
<b>Platform Designer Built-in Scripts</b>	Scripts that the Platform Designer tool provides. You cannot edit these scripts.
<b>User Scripts</b>	You can add your own scripts to this entry. Platform Designer saves these scripts to your user preference file, available in your home directory. The scripts that you add to this entry are available every time you open Platform Designer. Click <b>&lt;&lt;add script&gt;&gt;</b> to add a new script file to this entry. Double-click the <b>Description</b> field to add a description. Right-click the added script and click <b>Rename</b> to set a display name for the script.
<b>Project Scripts</b>	You can add your own scripts to this entry. Platform Designer saves these scripts to your current system. The scripts that you add to this entry are available only when you open this specific Platform Designer system. Click <b>&lt;&lt;add script&gt;&gt;</b> to add a new script file to this entry. Double-click the <b>Description</b> field to add a description or additional commands to the script. Right-click the added script and click <b>Rename</b> to set a display name for the script.
<b>Edit File</b>	Selecting the script in the <b>File</b> field displays the script in the pane below. Click <b>Edit File</b> to edit the script.
<b>Revert File</b>	Discards all your changes to the edited file.
<b>Save File</b>	Saves your changes to the edited file.
<b>Run Script</b>	Executes the selected script.
<b>System Scripting Messages</b>	Displays the warning and error messages when running the script.



Figure 172. System Scripting Tab



Note:

- To add additional commands to run before the script, right-click the column header and enable **Additional Commands**. Selecting this option displays a third column, in addition to **File** and **Description**. Double-click the entry in this field to add commands to execute before running your script. Alternatively, you can add the additional commands to your script, directly through the display pane in the middle, in the specified section.
- You can drag and drop items between the **Project Scripts** and **User Scripts** fields.

## 9.9 View and Filter Clock and Reset Domains in Your Platform Designer System

The Platform Designer clock and reset domains tabs allow you to see clock domains and reset domains in your Platform Designer system. Platform Designer determines clock and reset domains by the associated clocks and resets, which are displayed in tooltips for each interface in your system. You can filter your system to display particular components or interfaces within a selected clock or reset domain. The clock and reset domain tabs also provide quick access to performance bottlenecks by indicating connection points where Platform Designer automatically inserts clock crossing adapters and reset synchronizers during system generation. With these tools, you can more easily create optimal connections between interfaces.

Click **View > Clock Domains**, or **View > Reset Domains** to open the respective tabs in your workspace. The domain tools display as a tree with the current system at the root. You can select each clock or reset domain in the list to view associated interfaces.

When you select an element in the **Clock Domains** tab, the corresponding selection appears in the **System Contents** tab. You can select one or more single or multiple interfaces and modules. Mouse over tooltips in the **System Contents** tab to provide

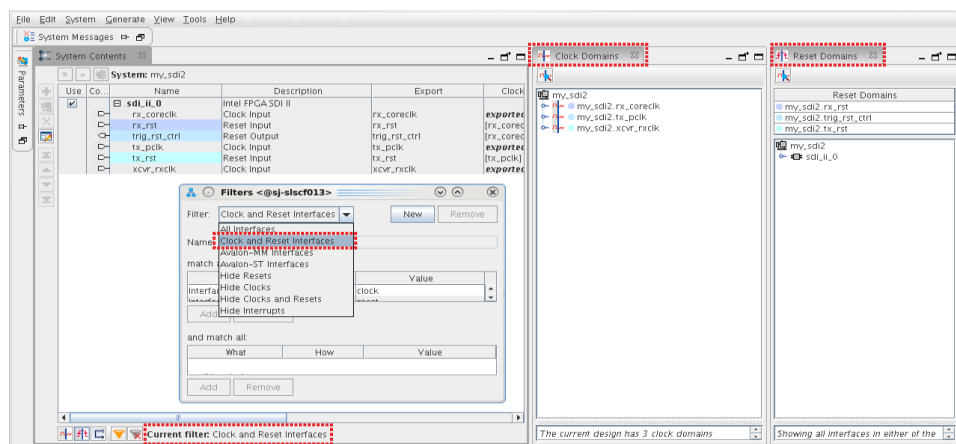
detailed information for all elements and connections. Colors that appear for the clocks and resets in the domain tools correspond to the colors in the **System Contents** and **Schematic** tabs.

Clock and reset control tools at the bottom on the **System Contents** tab allow you to toggle between highlighting clock or reset domains. You can further filter your view with options in the **Filters** dialog box, which is accessible by clicking the filter icon at the bottom of the **System Contents** tab. In the **Filters** dialog box, you can choose to view a single interface, or to hide clock, reset, or interrupt interfaces.

Clock and reset domain tools respond to global selection and edits, and help to provide answers to the following system design questions:

- How many clock and reset domains do you have in your Platform Designer system?
- What interfaces and modules does each clock or reset domain contain?
- Where do clock or reset crossings occur?
- At what connection points does Platform Designer automatically insert clock or reset adapters?
- Where do you have to manually insert a clock or reset adapter?

**Figure 173. Platform Designer Clock and Reset Domains**



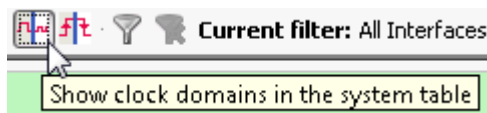
### 9.9.1 View Clock Domains in Your Platform Designer System

With the **Clock Domains** tab, you can filter the **System Contents** tab to display a single clock domain, or multiple clock domains. You can further filter your view with selections in the **Filters** dialog box. When you select an element in the **Clock Domains** tab, the corresponding selection appears highlighted in the **System Contents** tab.

1. To view clock domain interfaces and their connections in your Platform Designer system, click **View > Clock Domains** to open the Clock Domains tab.
2. To enable and disable highlighting of the clock domains in the **System Contents** tab, click the clock control tool at the bottom of the **System Contents** tab.

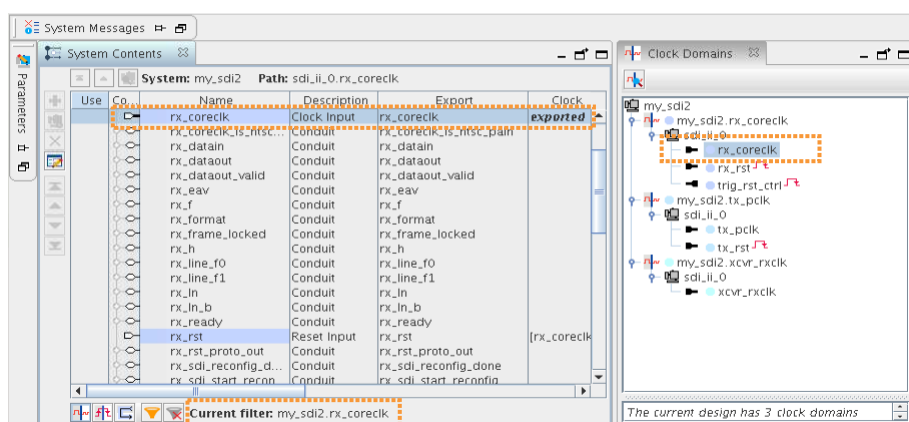


Figure 174. Clock Control Tool



- To view a single clock domain, or multiple clock domains and their modules and connections, click the clock name or names in the **Clock Domains** tab. The modules for the selected clock domain or domains and their connections appear highlighted in the **System Contents** tab. Detailed information for the current selection appears in the clock domain details pane. Red dots in the **Connections** column indicate auto insertions by Platform Designer during system generation, for example, a reset synchronizer or clock crossing adapter.

Figure 175. Clock Domains



- To view interfaces that cross clock domains, expand the **Clock Domain Crossings** icon in the **Clock Domains** tab, and select each element to view its details in the **System Contents** tab.

Platform Designer lists the interfaces that cross clock domain under **Clock Domain Crossings**. As you click through the elements, detailed information appears in the clock domain details pane. Platform Designer also highlights the selection in the **System Contents** tab.

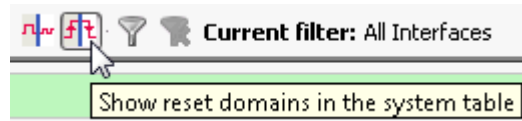
If a connection crosses a clock domain, the connection circle appears as a red dot in the **System Contents** tab. Mouse over tooltips at the red dot connections provide details about the connection, as well as what adapter type Platform Designer automatically inserts during system generation.

### 9.9.2 View Reset Domains in Your Platform Designer System

With the **Reset Domains** tab, you can filter the **System Contents** tab to display a single reset domain, or multiple reset domains. When you select an element in the **Reset Domains** tab, the corresponding selection appears in the **System Contents** tab.

- To view reset domain interfaces and their connections in your Platform Designer system, click **View > Reset Domains** to open the **Reset Domains** tab.
- To show reset domains in the **System Contents** tab, click the reset control tool at the bottom of the **System Contents** tab.

Figure 176. Reset Control Tool



- To view a single reset domain, or multiple reset domains and their modules and connections, click the reset names in the **Reset Domain** tab.

Platform Designer displays your selection according to the following rules:

- When you select multiple reset domains, the **System Contents** tab shows interfaces and modules in both reset domains.
- When you select a single reset domain, the other reset domains are grayed out, unless the two domains have interfaces in common.
- Reset interfaces appear black when connected to multiple reset domains.
- Reset interfaces appear gray when they are not connected to all of the selected reset domains.
- If an interface is contained in multiple reset domains, the interface is grayed out.

Detailed information for your selection appears in the reset domain details pane.

*Note:* Red dots in the **Connections** column between reset sinks and sources indicate auto insertions by Platform Designer during system generation, for example, a reset synchronizer. Platform Designer decides when to display a red dot with the following protocol, and ends the decision process at first match.

- Multiple resets fan into a common sink.
- Reset inputs are associated with different clock domains.
- Reset inputs have different synchronicity.

### 9.9.3 Filter Platform Designer Clock and Reset Domains in the System Contents Tab

You can filter the display of your Platform Designer clock and reset domains in the **System Contents** tab.

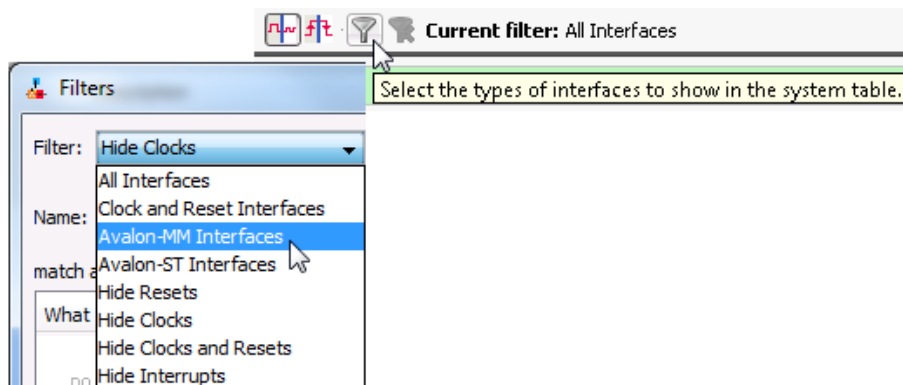
- To filter the display in the **System Contents** tab to view only a particular interface and its connections, or to choose to hide clock, reset, or interrupt interfaces, click the **Filters** icon in the clock and reset control tool to open the **Filters** dialog box.

The selected interfaces appear in the **System Contents** tab.



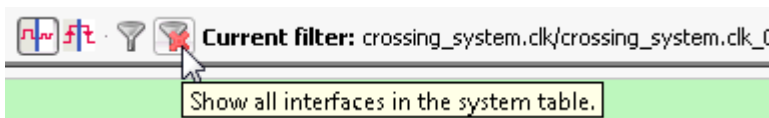


Figure 177. Filters Dialog Box



2. To clear all clock and reset filters in the **System Contents** tab and show all interfaces, click the **Filters** icon with the red "x" in the clock and reset control tool.

Figure 178. Show All Interfaces



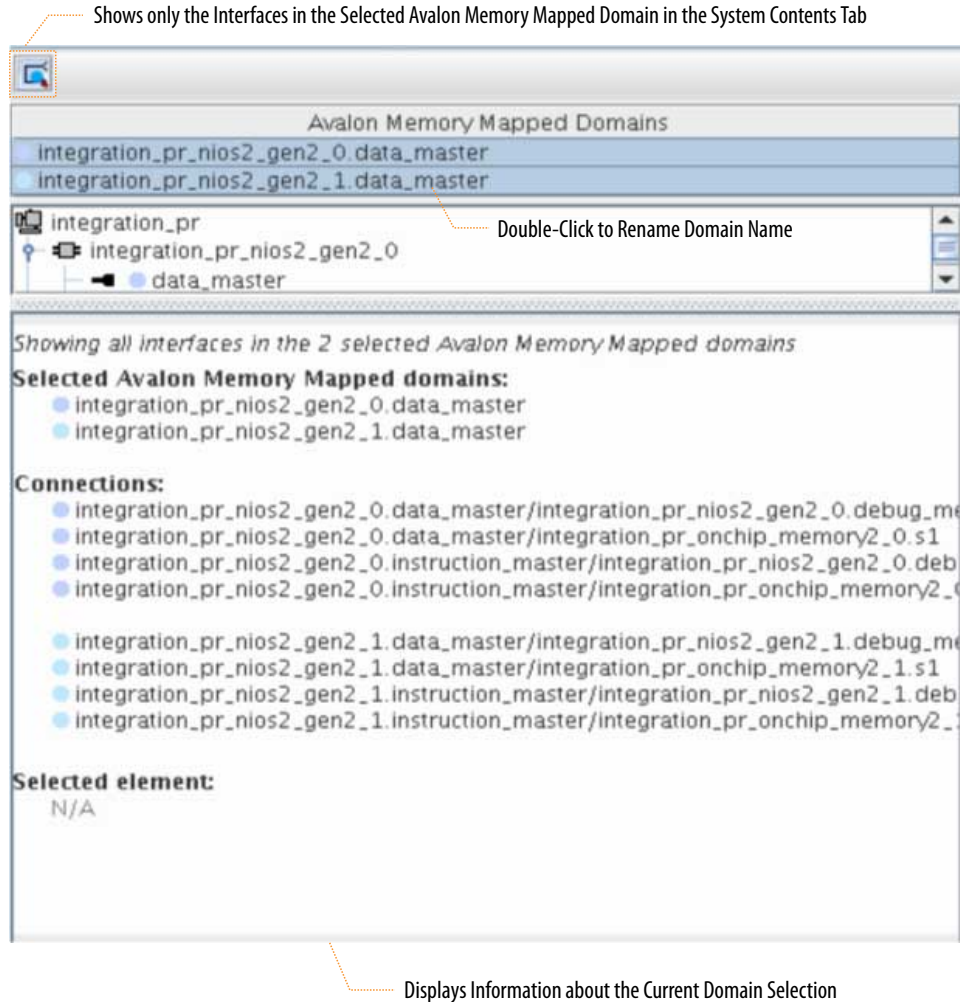
### 9.9.4 View Avalon Memory Mapped Domains in Your Platform Designer System

The **Avalon Memory Mapped Domains** tab (**View > Avalon Memory Mapped Domains**) displays a list of all the Avalon domains in the system.

With the **Avalon Memory Mapped Domains** tab, you can filter the **System Contents** tab to display a single Avalon domain, or multiple domains. You can further filter your view with selections in the **Filters** dialog box. When you select a domain in the **Avalon Memory Mapped Domains** tab, the corresponding selection is highlighted in the **System Contents** tab.

To rename an Avalon memory mapped domain, double-click the domain name. Detailed information for the current selection appears in the Avalon domain details pane. Also, you can choose to view only the selected domain's interfaces in the **System Contents** tab.

**Figure 179. Avalon Memory Mapped Domains Tab**



To enable and disable the highlighting of the Avalon domains in the **System Contents** tab, click the domain control tool at the bottom of the **System Contents** tab.

**Figure 180. Avalon Memory Mapped Domains Control Tool**

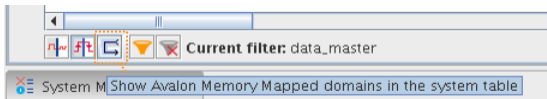
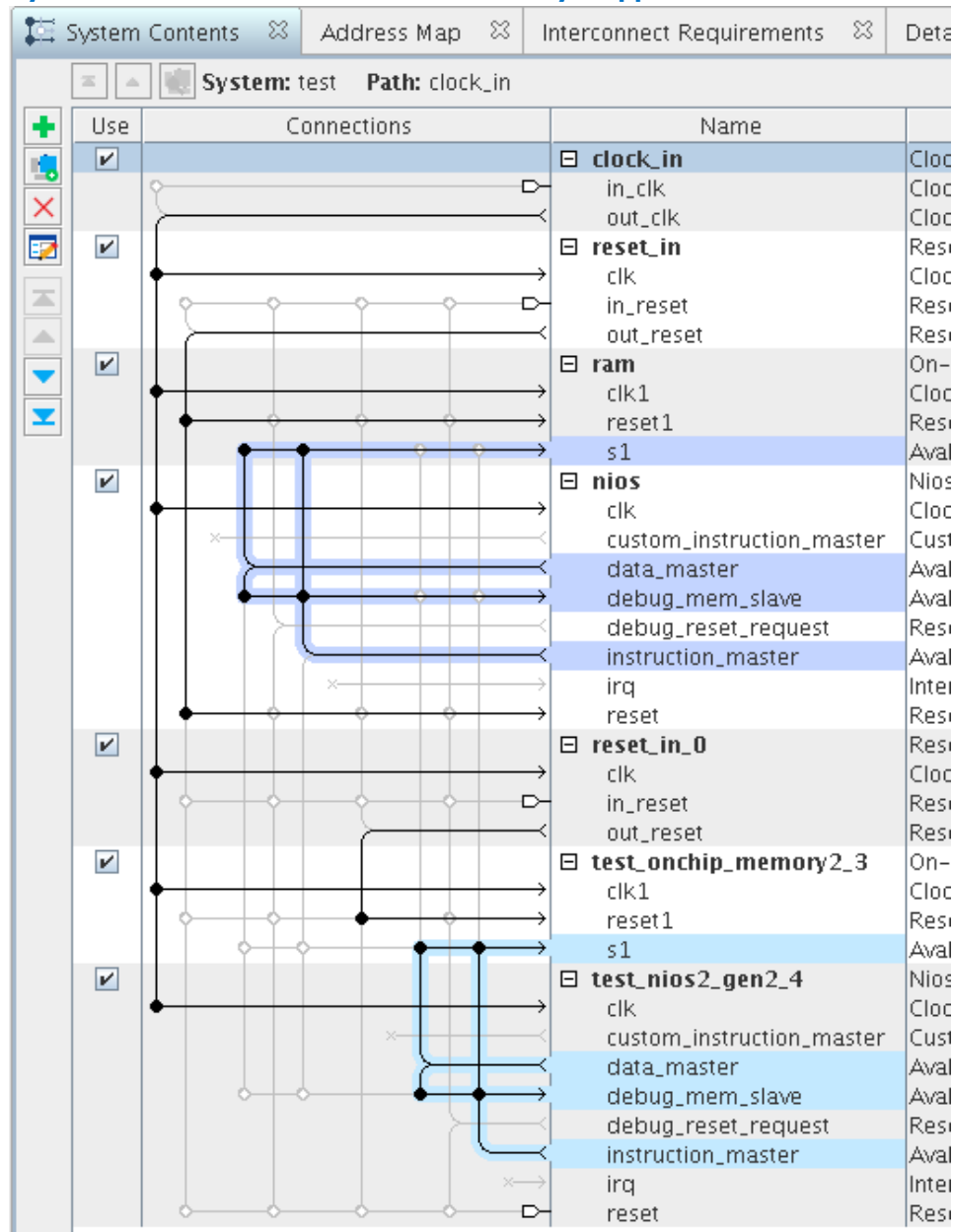




Figure 181. System Contents Tab with Avalon Memory Mapped Domains Selected



## 9.10 Specify Platform Designer Interconnect Requirements

The **Interconnect Requirements** tab allows you to apply system-wide, `$system`, and interface interconnect requirements for IP components in your system. Options in the **Setting** column vary depending on what you select in the **Identifier** column. Click the drop-down menu to select the settings, and to assign the corresponding values to the settings.

**Table 98. Specifying System-Wide Interconnect Requirements**

Option	Description
<b>Limit interconnect pipeline stages to</b>	Specifies the maximum number of pipeline stages that Platform Designer may insert in each command and response path to increase the $f_{MAX}$ at the expense of additional latency. You can specify between 0–4 pipeline stages, where 0 means that the interconnect has a combinational datapath. Choosing 3 or 4 pipeline stages may significantly increase the logic utilization of the system. This setting is specific for each Platform Designer system or subsystem, meaning that each subsystem can have a different setting. Additional latency is added once on the command path, and once on the response path. You can manually adjust this setting in the <b>Memory-Mapped Interconnect</b> tab. Access this tab by clicking <b>Show System With Platform Designer Interconnect command</b> on the <b>System</b> menu.
<b>Clock crossing adapter type</b>	Specifies the default implementation for automatically inserted clock crossing adapters: <ul style="list-style-type: none"> <li>• <b>Handshake</b>—This adapter uses a simple hand-shaking protocol to propagate transfer control signals and responses across the clock boundary. This methodology uses fewer hardware resources because each transfer is safely propagated to the target domain before the next transfer can begin. The <b>Handshake</b> adapter is appropriate for systems with low throughput requirements.</li> <li>• <b>FIFO</b>—This adapter uses dual-clock FIFOs for synchronization. The latency of the FIFO-based adapter is a couple of clock cycles more than the handshaking clock crossing component. However, the FIFO-based adapter can sustain higher throughput because it supports multiple transactions at any given time. FIFO-based clock crossing adapters require more resources. The <b>FIFO</b> adapter is appropriate for memory-mapped transfers requiring high throughput across clock domains.</li> <li>• <b>Auto</b>—If you select <b>Auto</b>, Platform Designer specifies the <b>FIFO</b> adapter for bursting links, and the <b>Handshake</b> adapter for all other links.</li> </ul>
<b>Automate default slave insertion</b>	Specifies whether you want Platform Designer to automatically insert a default slave for undefined memory region accesses during system generation.
<b>Enable instrumentation</b>	When you set this option to TRUE, Platform Designer enables debug instrumentation in the Platform Designer interconnect, which then monitors interconnect performance in the system console.
<b>Burst Adapter Implementation</b>	Allows you to choose the converter type that Platform Designer applies to each burst. <ul style="list-style-type: none"> <li>• <b>Generic converter (slower, lower area)</b>—Default. Controls all burst conversions with a single converter that is able to adapt incoming burst types. This results in an adapter that has lower <math>f_{MAX}</math>, but smaller area.</li> <li>• <b>Per-burst-type converter (faster, higher area)</b>—Controls incoming bursts with a particular converter, depending on the burst type. This results in an adapter that has higher <math>f_{MAX}</math>, but higher area. This setting is useful when you have AXI masters or slaves and you want a higher <math>f_{MAX}</math>.</li> </ul>
<b>Enable ECC protection</b>	Specifies the default implementation for ECC protection for memory elements. Currently supports only Read Data FIFO ( <code>rdata_fifo</code> ) instances.. <ul style="list-style-type: none"> <li>• <b>FALSE</b>—Default. ECC protection is disabled for memory elements in the Platform Designer interconnect.</li> <li>• <b>TRUE</b>—ECC protection is enabled for memory elements. Platform Designer interconnect sends ECC errors that cannot be corrected as <code>DECODEERROR (DECERR)</code> on the Avalon response bus. This setting may increase logic utilization and cause lower <math>f_{MAX}</math>, but provides additional protection against data corruption.</li> </ul> <p><i>Note: For more information about Error Correction Coding (ECC), refer to <a href="#">Error Correction Coding in Platform Designer Interconnect</a>.</i></p>
<b>Interconnect type</b>	Allows you to select the implementation of Platform Designer interconnect. You can select one of the following options: <ul style="list-style-type: none"> <li>• <b>Standard</b>—suitable for all devices</li> <li>• <b>(Alpha release) Hyperflex-optimized</b>—suitable for latency-tolerant Intel Stratix 10 applications. This option has higher potential <math>f_{max}</math> and bandwidth, at the expense of increased latency</li> </ul>

**Table 99. Specifying Interface Interconnect Requirements**

You can apply the following interconnect requirements when you select a component interface as the **Identifier** in the **Interconnect Requirements** tab, in the **All Requirements** table.

Option	Value	Description
<b>Security</b>	<ul style="list-style-type: none"> <li>Non-secure</li> <li>Secure</li> <li>Secure ranges</li> <li>TrustZone-aware</li> </ul>	After you establish connections between the masters and slaves, allows you to set the security options, as needed, for each master and slave in your system. <i>Note:</i> You can also set these values in the <b>Security</b> column in the <b>System Contents</b> tab.
<b>Secure address ranges</b>	Accepts valid address range.	Allows you to type in any valid address range.

### Related Links

[Error Correction Coding \(ECC\) in Platform Designer Interconnect](#) on page 718

## 9.11 Manage Platform Designer System Security

Arm TrustZone is the security extension of the Arm-based architecture. It includes secure and non-secure transactions designations, and a protocol for processing between the designations. TrustZone security support is a part of the Platform Designer interconnect.

The AXI `AxPROT` protection signal specifies a secure or non-secure transaction. When an AXI master sends a command, the `AxPROT` signal specifies whether the command is secure or non-secure. When an AXI slave receives a command, the `AxPROT` signal determines whether the command is secure or non-secure. Determining the security of a transaction while sending or receiving a transaction is a run-time protocol.

The Avalon specification does not include a protection signal as part of its specification. When an Avalon master sends a command, it has no embedded security and Platform Designer recognizes the command as non-secure. When an Avalon slave receives a command, it also has no embedded security, and the slave always accepts the command and responds.

AXI masters and slaves can be TrustZone-aware. All other master and slave interfaces, such as Avalon-MM interfaces, are non-TrustZone-aware. You can set compile-time security support for all components (except AXI masters, including AMBA 3 AXI, AMBA 3 AXI-Lite) in the **Security** column in the **System Contents** tab, or in the **Interconnect Requirements** tab under the **Identifier** column for the master or slave interface. To begin creating a secure system, you must first add masters and slaves to your system, and the connections between them. After you establish connections between the masters and slaves, you can then set the security options, as needed.

An example of when you may need to specify compile-time security support is when an Avalon master needs to communicate with a secure AXI slave, and you can specify whether the connection point is secure or non-secure. You can specify a compile-time secure address ranges for a memory slave if an interface-level security setting is not sufficient.

### Related Links

- [Platform Designer Interconnect](#) on page 659
- [Platform Designer System Design Components](#) on page 914

### 9.11.1 Configure Platform Designer Security Settings Between Interfaces

The AXI AxPROT signal specifies a transaction as secure or non-secure at runtime when a master sends a transaction. Platform Designer identifies AXI master interfaces as TrustZone-aware. You can configure AXI slaves as TrustZone-aware, secure, non-secure, or secure ranges.

**Table 100. Compile-Time Security Options**

For non-TrustZone-aware components, compile-time security support options are available in Platform Designer on the **System Contents** tab, or on the **Interconnect Requirements** tab.

Compile-Time Security Options	Description
<b>Non-secure</b>	Master sends only non-secure transactions, and the slave receives any transaction, secure or non-secure.
<b>Secure</b>	Master sends only secure transactions, and the slave receives only secure transactions.
<b>Secure ranges</b>	Applies to only the slave interface. The specified address ranges within the slave's address span are secure, all other address ranges are not. The format is a comma-separated list of inclusive-low and inclusive-high addresses, for example, 0x0:0xffff, 0x2000:0x20fff.

After setting compile-time security options for non-TrustZone-aware master and slave interfaces, you must identify those masters that require a default slave before generation. To designate a slave interface as the default slave, turn on **Default Slave** in the **System Contents** tab. A master can have only one default slave.

*Note:* The **Security** and **Default Slave** columns in the **System Contents** tab are hidden by default. Right-click the **System Contents** header to select which columns you want to display.

The following are descriptions of security support for master and slave interfaces. These description can guide you in your design decisions when you want to create secure systems that have mixed secure and non-TrustZone-aware components:

- All AXI, AMBA 3 AXI, and AMBA 3 AXI-Lite masters are TrustZone-aware.
- You can set AXI, AMBA 3 AXI, and AMBA 3 AXI-Lite slaves as TrustZone-aware, secure, non-secure, or secure range ranges.
- You can set non-AXI master interfaces as secure or non-secure.
- You can set non-AXI slave interfaces as secure, non-secure, or secure address ranges.

### 9.11.2 Specify a Default Slave in a Platform Designer System

If a master issues "per-access" or "not allowed" transactions, your design must contain a default slave. Per-access refers to the ability of a TrustZone-aware master to allow or disallow access or transactions. A transaction that violates security is rerouted to the default slave and subsequently responds to the master with an error. You can designate any slave as the default slave.

You can share a default slave between multiple masters. You should have one default slave for each interconnect domain. An interconnect domain is a group of connected memory-mapped masters and slaves that share the same interconnect. The altera\_error\_response\_slave component includes the required TrustZone features.



You can achieve an optimized secure system by partitioning your design and carefully designating secure or non-secure address maps to maintain reliable data. Avoid a design where, under the same hierarchy, a non-secure master initiates transactions to a secure slave resulting in unsuccessful transfers.

**Table 101. Secure and Non-Secure Access Between Master, Slave, and Memory Components**

Transaction Type	TrustZone-aware Master	Non-TrustZone-aware Master Secure	Non-TrustZone-aware Master Non-Secure
TrustZone-aware slave/memory	OK	OK	OK
Non-TrustZone-aware slave (secure)	Per-access	OK	Not allowed
Non-TrustZone-aware slave (non-secure)	OK	OK	OK
Non-TrustZone-aware memory (secure region)	Per-access	OK	Not allowed
Non-TrustZone-aware memory (non-secure region)	OK	OK	OK

#### Related Links

- [Error Response Slave](#) on page 937
- [Designating a Default Slave in the System Contents Tab](#) on page 942

### 9.11.3 Access Undefined Memory Regions

When a transaction from a master targets a memory region that is not specified in the slave memory map, it is known as an *access to an undefined memory region*. To ensure predictable response behavior when this occurs, you must add a default slave to your design. Platform Designer then routes undefined memory region accesses to the default slave, which terminates the transaction with an error response.

You can designate any memory-mapped slave as a default slave. Intel recommends that you have only one default slave for each interconnect domain in your system. Accessing undefined memory regions can occur in the following cases:

- When there are gaps within the accessible memory map region that are within the addressable range of slaves, but are not mapped.
- Accesses by a master to a region that does not belong to any slaves that is mapped to the master.
- When a non-secured transaction is accessing a secured slave. This applies to only slaves that are secured at compilation time.
- When a read-only slave is accessed with a write command, or a write-only slave is accessed with a read command.

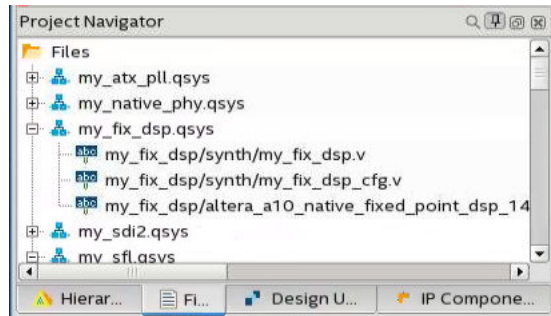
To designate a slave as the default slave, for the selected component, turn on **Default Slave** in the **Systems Content** tab.

*Note:* If you do not specify the default slave, Platform Designer automatically assigns the slave at the lowest address within the memory map for the master that issues the request as the default slave.

## 9.12 Integrating a Platform Designer System with a Intel Quartus Prime Project

The Intel Quartus Prime software tightly links with Platform Designer system creation. Platform Designer requires you to specify a Intel Quartus Prime project at time of system creation. The Intel Quartus Prime software automatically adds all .qsys and all .ip files for the associated Platform Designer system to your Intel Quartus Prime project. When you open your Intel Quartus Prime project, the project automatically lists all the files related to the Platform Designer system.

Figure 182. Platform Designer System Files in Intel Quartus Prime Project



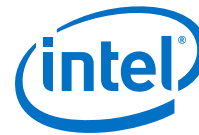
## 9.13 Manage IP Settings in the Intel Quartus Prime Software

To specify the following IP Settings in the Intel Quartus Prime software, click **Tools** ► **Option** ► **IP Settings**:

Table 102. IP Settings

Setting	Description
<b>Maximum Platform Designer memory usage</b>	Allows you to increase memory usage for Platform Designer if you experience slow processing for large systems, or if Platform Designer reports an <b>Out of Memory</b> error.
<b>IP generation HDL preference</b>	The Intel Quartus Prime software uses this setting when the .qsys file appears in the <b>Files</b> list for the current project in the <b>Settings</b> dialog box and you run Analysis & Synthesis. Platform Designer uses this setting when you generate HDL files.
<b>Automatically add Intel Quartus Prime IP files to all projects</b>	The Intel Quartus Prime software uses this setting when you create an IP core file variation with options in the Intel Quartus Prime IP Catalog and parameter editor. When turned on, the Intel Quartus Prime software adds the IP variation files to the project currently open.
<b>IP Catalog Search Locations</b>	The Intel Quartus Prime software uses the settings that you specify for global and project search paths under <b>IP Search Locations</b> , to populate the Intel Quartus Prime software IP Catalog.  Platform Designer uses the settings that you specify for global search paths under <b>IP Search Locations</b> to populate the Platform Designer IP Catalog, which appears in Platform Designer ( <b>Tools</b> ► <b>Options</b> ). Platform Designer uses the project search path settings to populate the Platform Designer IP Catalog when you open Platform Designer from within the Intel Quartus Prime software ( <b>Tools</b> ► <b>Platform Designer</b> ), but not when you open Platform Designer from the command-line.





*Note:* You can also access **IP Settings** by clicking **Assignments > Settings > IP Settings**. This access is available only when you have a Intel Quartus Prime project open. This allows you access to **IP Settings** when you want to create IP cores independent of a Intel Quartus Prime project. Settings that you apply or create in either location are shared.

### 9.13.1 Opening Platform Designer with Additional Memory

If your Platform Designer system requires more than the 512 megabytes of default memory, you can increase the amount of memory either in the Intel Quartus Prime software **Options** dialog box, or at the command-line.

- When you open Platform Designer from within the Intel Quartus Prime software, you can increase memory for your Platform Designer system, by clicking **Tools > Options > IP Settings**, and then selecting the appropriate amount of memory with the **Maximum Platform Designer memory usage** option.
- When you open Platform Designer from the command-line, you can add an option to increase the memory. For example, the following `qsys-edit` command allows you to open Platform Designer with 1 gigabytes of memory.

```
qsys-edit --jvm-max-heap-size=1g
```

## 9.14 Generate a Platform Designer System

In Platform Designer, you can choose options for generation of synthesis, simulation and testbench files for your Platform Designer system.

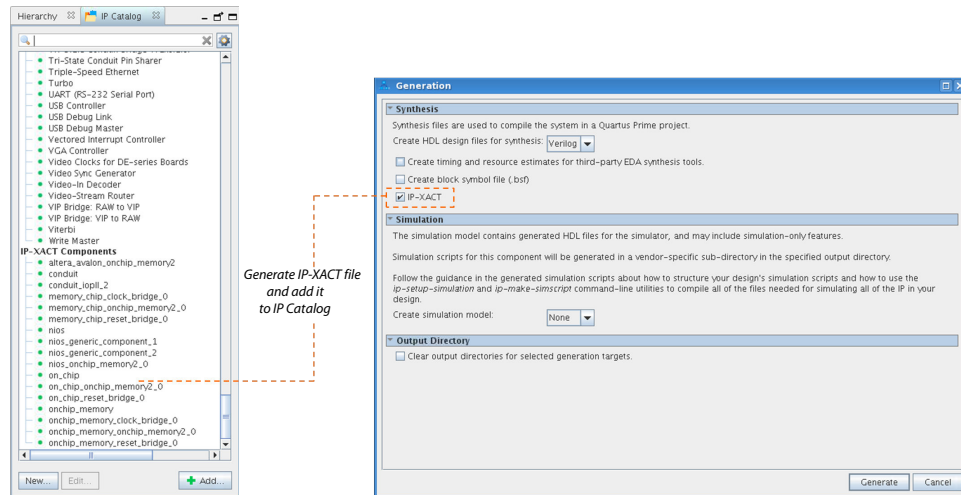
Platform Designer system generation creates the interconnect between IP components and generates synthesis and simulation HDL files. You can generate a testbench system that adds Bus Functional Models (BFMs) that interact with your system in a simulator.

When you make changes to a system, Platform Designer gives you the option to exit without generating. If you choose to generate your system before you exit, the **Generation** dialog box opens and allows you to select generation options.

The **Generate HDL** button in the lower-right of the Platform Designer window allows you to quickly generate synthesis and simulation files for your system.

*Note:* If you cannot find the memory interface generated by Platform Designer when you use EMIF (External Memory Interface Debug Toolkit), verify that the `.sopcinfo` file appears in your Platform Designer project folder.

Figure 183. Generating IP-XACT file for the system



### Related Links

- [Avalon Verification IP Suite User Guide](#)
- [Mentor Verification IP \(VIP\) Altera Edition \(AE\)](#)
- [External Memory Interface Debug Toolkit](#)

#### 9.14.1 Set the Generation ID

The **Generation Id** parameter is a unique integer value that is set to a timestamp during Platform Designer system generation. System tools, such as Nios II or HPS (Hard Processor System) use the **Generation ID** to ensure software-build compatibility with your Platform Designer system.

To set the **Generation Id** parameter, select the top-level system in the **Hierarchy** tab, and then locating the parameter in the open **Parameters** tab.

#### 9.14.2 Generate Files for Synthesis and Simulation

Platform Designer generates files for synthesis in Intel Quartus Prime software and simulation in a third-party simulator.

In Platform Designer, you can generate simulation HDL files (**Generate ► Generate HDL**), which can include simulation-only features targeted towards your simulator. You can generate simulation files as Verilog or VHDL.

*Note:* For a list of Intel-supported simulators, refer to *Simulating Intel Designs*.

Platform Designer supports standard and legacy device generation. Standard device generation refers to generating files for the Intel Arria 10 and later device families. Legacy device generation refers to generating files for device families prior to the release of the Intel Arria 10 device family, including MAX 10 devices.



The **Output Directory** option applies to both synthesis and simulation generation. By default, the path of the generation output directory is fixed relative to the `.qsys` file. You can change the default directory in the **Generation** dialog box for legacy devices. For standard devices, the generation directory is fixed to the Platform Designer project directory.

*Note:* If you need to change top-level I/O pin or instance names, create a top-level HDL file that instantiates the Platform Designer system. The Platform Designer-generated output is then instantiated in your design without changes to the Platform Designer-generated output files.

The following options in the **Generation** dialog box (**Generate** ► **Generate HDL**) allow you to generate synthesis and simulation files:

**Table 103. Generation Dialog Box Options**

Option	Description
<b>Create HDL design files for synthesis</b>	Generates Verilog HDL or VHDL design files for the system's top-level definition and child instances for the selected target language. Synthesis file generation is optional.
<b>Create timing and resource estimates for third-party EDA synthesis tools</b>	Generates a non-functional Verilog Design File ( <code>.v</code> ) for use by some third-party EDA synthesis tools. Estimates timing and resource usage for your IP component. The generated netlist file name is <code>&lt;your_ip_component_name&gt;_syn.v</code> .
<b>Create Block Symbol File (.bsf)</b>	Allows you to optionally create a ( <code>.bsf</code> ) file to use in a schematic Block Diagram File ( <code>.bdf</code> ).
<b>IP-XACT</b>	Generates an IP-XACT file for the system, and adds the file to the IP Catalog.
<b>Create simulation model</b>	Allows you to optionally generate Verilog HDL or VHDL simulation model files, and simulation scripts.
<b>Clear output directories for selected generation targets</b>	Clears previous generation attempts for current synthesis or simulation.

*Note:* ModelSim - Intel FPGA Edition now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation. Therefore, Intel simulation libraries may not be compatible with single language simulators. If you have a VHDL-only license, some versions of Mentor simulators may not be able to simulate IP written in Verilog. As a workaround, you can use ModelSim - Intel FPGA Edition, or purchase a mixed language simulation license from Mentor.

### Related Links

[Simulating Intel Designs](#)

## 9.14.2.1 Files Generated for Intel FPGA IP Cores and Platform Designer Systems

The Intel Quartus Prime Pro Edition software generates the following output file structure for IP cores and Platform Designer systems. The Intel Quartus Prime Pro Edition Platform Designer software automatically adds the generated `.ip` and `.qsys` files to your Intel Quartus Prime project.

Figure 184. Files generated for IP cores and Platform Designer Systems

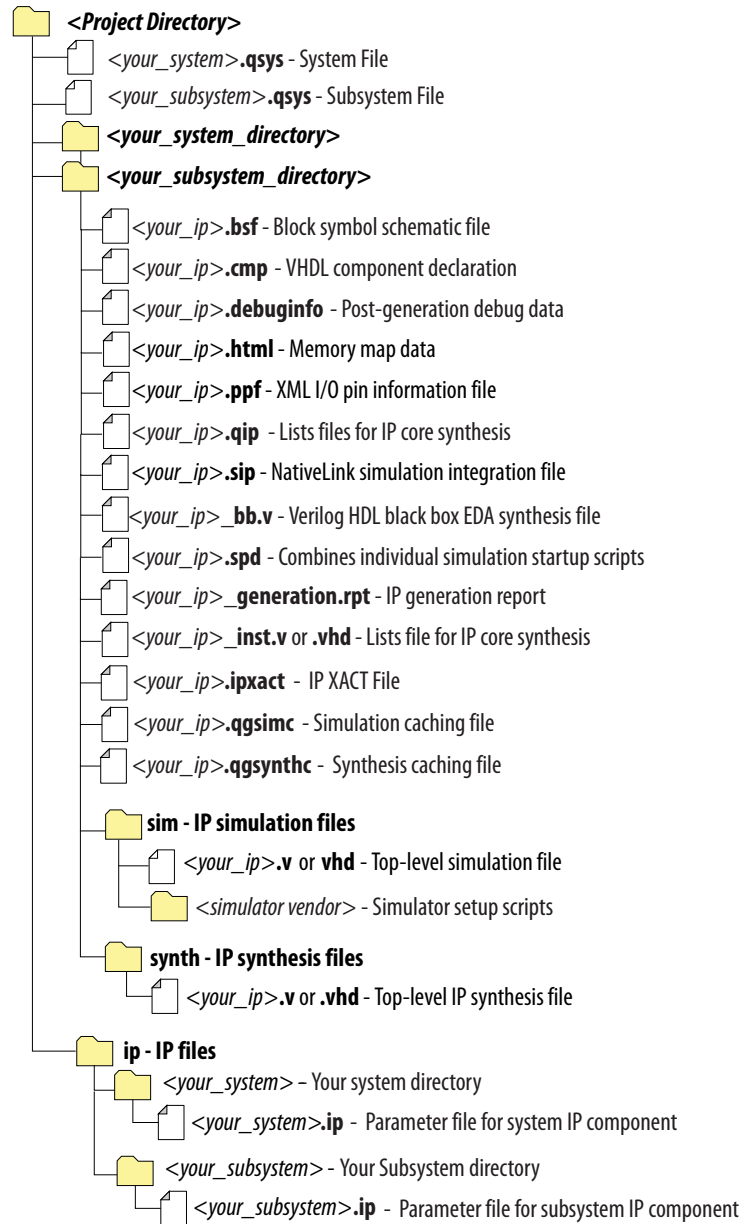
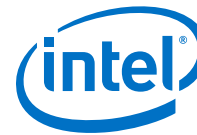


Table 104. IP Core and Platform Designer Simulation Files

File Name	Description
<my_system>.qsys	The Platform Designer system.
<my_subsystem>.qsys	The Platform Designer subsystem.
ip/	Contains the parameter files for the IP components in the system and subsystem(s).
<i>continued...</i>	



File Name	Description
<my_ip>.cmp	The VHDL Component Declaration ( <b>.cmp</b> ) file is a text file that contains local generic and port definitions that you can use in VHDL design files.
<my_ip>_generation.rpt	IP or Platform Designer generation log file. A summary of the messages during IP generation.
<my_ip>.qgsimc	Simulation caching file that compares the <b>.qsys</b> and <b>.ip</b> files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<my_ip>.qgsynth	Synthesis caching file that compares the <b>.qsys</b> and <b>.ip</b> files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<my_ip>.qip	Contains all the required information about the IP component to integrate and compile the IP component in the Intel Quartus Prime software.
<my_ip>.csv	Contains information about the upgrade status of the IP component.
.bsf	A Block Symbol File ( <b>.bsf</b> ) representation of the IP variation for use in Block Diagram Files (<my_ip>. <b>bdf</b> ).
<my_ip<>.spd	Required input file for <b>ip-make-simscript</b> to generate simulation scripts for supported simulators. The <b>.spd</b> file contains a list of files generated for simulation, along with information about memories that you can initialize.
<my_ip>.ppf	The Pin Planner File ( <b>.ppf</b> ) stores the port and node assignments for IP components created for use with the Pin Planner.
<my_ip>_bb.v	Use the Verilog black box ( <b>_bb.v</b> ) file as an empty module declaration for use as a black box.
<my_ip>.sip	Contains information required for NativeLink simulation of IP components. Add the <b>.sip</b> file to your Intel Quartus Prime Standard Edition project to enable NativeLink for supported devices. The Intel Quartus Prime Pro Edition software does not support NativeLink simulation.
<my_ip>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<my_ip>.regmap	If the IP contains register information, the Intel Quartus Prime software generates the <b>.regmap</b> file. The <b>.regmap</b> file describes the register map information of master and slave interfaces. This file complements the <b>.sopcinfo</b> file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<my_ip>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Platform Designer system. During synthesis, the Intel Quartus Prime software stores the <b>.svd</b> files for slave interface visible to the System Console masters in the <b>.sof</b> file in the debug session. System Console reads this section, which Platform Designer can query for register map information. For system slaves, Platform Designer can access the registers by name.
<my_ip>.v <my_ip>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a ModelSim script <b>msim_setup.tcl</b> to set up and run a simulation.
aldec/	Contains a Riviera-PRO script <b>rivierapro_setup.tcl</b> to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script <b>vcs_setup.sh</b> to set up and run a VCS simulation. Contains a shell script <b>vcsmx_setup.sh</b> and <b>synopsys_ sim.setup</b> file to set up and run a VCS MX® simulation.

*continued...*



File Name	Description
/cadence	Contains a shell script <code>ncsim_setup.sh</code> and other setup files to set up and run an NCSIM simulation.
/submodules	Contains HDL files for the IP core submodule.
<IP submodule>/	For each generated IP submodule directory, Platform Designer generates /synth and /sim sub-directories.

### 9.14.3 Generate Files for a Testbench Platform Designer System

Platform Designer testbench is a new system that instantiates the current Platform Designer system by adding BFM to drive the top-level interfaces. BFM interact with the system in the simulator. You can use options in the **Generation** dialog box (**Generate > Generate Testbench System**) to generate a testbench Platform Designer system.

You can generate a standard or simple testbench system with BFM or Mentor Verification IP (for AMBA 3 AXI or AMBA 3 AXI) IP components that drive the external interfaces of your system. Platform Designer generates a Verilog HDL or VHDL simulation model for the testbench system to use in your simulation tool. You should first generate a testbench system, and then modify the testbench system in Platform Designer before generating its simulation model. In most cases, you should select only one of the simulation model options.

By default, the path of the generation output directory is fixed relative to the `.qsys` file. You can change the default directory in the **Generation** dialog box for legacy devices. For standard devices, the generation directory is fixed to the Platform Designer project directory.

The following options are available for generating a Platform Designer testbench system:

Option	Description
<b>Create testbench Platform Designer system</b>	<ul style="list-style-type: none"> <li><b>Standard, BFM for standard Platform Designer Interconnect</b>—Creates a testbench Platform Designer system with BFM IP components attached to exported Avalon and AMBA 3 AXI or AMBA 3 AXI interfaces. Includes any simulation partner modules specified by IP components in the system. The testbench generator supports AXI interfaces and can connect AMBA 3 AXI or AMBA 3 AXI interfaces to Mentor Graphics AMBA 3 AXI or AMBA 3 AXI master/slave BFM. However, BFM support address widths only up to 32-bits.</li> <li><b>Simple, BFM for clocks and resets</b>—Creates a testbench Platform Designer system with BFM IP components driving only clock and reset interfaces. Includes any simulation partner modules specified by IP components in the system.</li> </ul>
<b>Create testbench simulation model</b>	Creates Verilog HDL or VHDL simulation model files and simulation scripts for the testbench Platform Designer system currently open in your workspace. Use this option if you do not need to modify the Platform Designer-generated testbench before running the simulation.

**Note:** ModelSim - Intel FPGA Edition now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation. Therefore, Intel simulation libraries may not be compatible with single language simulators. If you have a VHDL-only license, some versions of Mentor simulators may not be able to simulate IP written in Verilog. As a workaround, you can use ModelSim - Intel FPGA Edition, or purchase a mixed language simulation license from Mentor.



### 9.14.3.1 Files Generated for Platform Designer Testbench

**Table 105. Platform Designer-Generated Testbench Files**

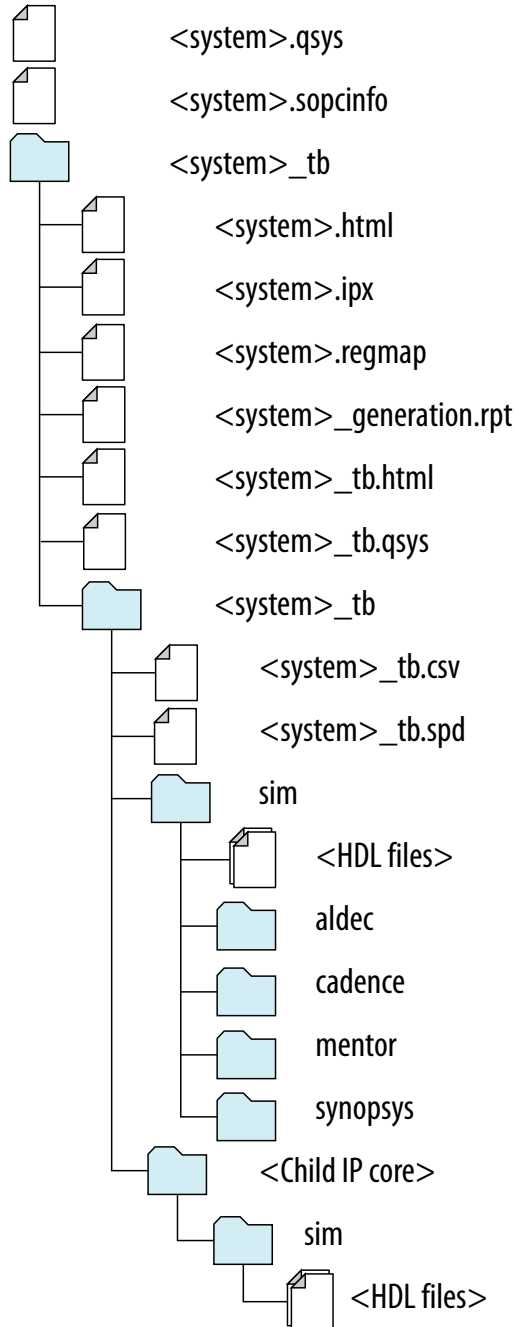
File Name or Directory Name	Description
<system>_tb.qsys	The Platform Designer testbench system.
<system>_tb.v or <system>_tb.vhd	The top-level testbench file that connects BFM to the top-level interfaces of <system>_tb.qsys.
<system>_tb.spd	Required input file for ip-make-simscript to generate simulation scripts for supported simulators. The .spd file contains a list of files generated for simulation and information about memory that you can initialize.
<system>.html and <system>_tb.html	A system report that contains connection information, a memory map showing the address of each slave with respect to each master to which it is connected, and parameter assignments.
<system>_generation.rpt	Platform Designer generation log file. A summary of the messages that Platform Designer issues during testbench system generation.
<system>.ipx	The IP Index File (.ipx) lists the available IP components, or a reference to other directories to search for IP components.
<system>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Platform Designer system. Similarly, during synthesis the .svd files for slave interfaces visible to System Console masters are stored in the .sof file in the debug section. System Console reads this section, which Platform Designer can query for register map information. For system slaves, Platform Designer can access the registers by name.
mentor/	Contains a ModelSim script msim_setup.tcl to set up and run a simulation
aldec/	Contains a Riviera-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS simulation. Contains a shell script vcsmx_setup.sh and synopsys_ sim.setup file to set up and run a VCS MX simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules	Contains HDL files for the submodule of the Platform Designer testbench system.
<child IP cores>/	For each generated child IP core directory, Platform Designer testbench generates /synth and /sim subdirectories.

### 9.14.3.2 Platform Designer Testbench Simulation Output Directories

The /sim and /simulation directories contain the Platform Designer-generated output files to simulate your Platform Designer testbench system.

Figure 185. Platform Designer Simulation Testbench Directory Structure

Output Directory Structure



### 9.14.3.3 Generate and Modify a Platform Designer Testbench System

You can use the following steps to create a Platform Designer testbench system of your Platform Designer system.





1. Create a Platform Designer system.
2. Generate a testbench system in the Platform Designer **Generation** dialog box (**Generate** > **Generate Testbench System**).
3. Open the testbench system in Platform Designer. Make changes to the BFM, as needed, such as changing the instance names and **VHDL ID** value. For example, you can modify the **VHDL ID** value in the **Altera Avalon Interrupt Source IP** component.
4. If you modify a BFM, regenerate the simulation model for the testbench system.
5. Create a custom test program for the BFM.
6. Compile and load the Platform Designer system and testbench into your simulator, and then run the simulation.

### 9.14.4 Platform Designer Simulation Scripts

Platform Designer generates simulation scripts to set up the simulation environment for Mentor Graphics Modelsim and Questasim, Synopsys VCS and VCS MX, Cadence Incisive Enterprise Simulator® (NCSIM), and the Aldec Riviera-PRO Simulator.

Platform Designer generates simulation scripts for all `.ip` and `.qsys` files of a system and places the files in the simulation script output folder (`<top-level system name>/sim/<simulator name>`).

Platform Designer always generates the simulation scripts from the currently loaded system down. You can open a subsystem and choose to generate a simulation script just for that subsystem.

You can use scripts to compile the required device libraries and system design files in the correct order and elaborate or load the top-level system for simulation.

**Table 106. Simulation Script Variables**

The simulation scripts provide variables that allow flexibility in your simulation environment.

Variable	Description
TOP_LEVEL_NAME	If the testbench Platform Designer system is not the top-level instance in your simulation environment because you instantiate the Platform Designer testbench within your own top-level simulation file, set the TOP_LEVEL_NAME variable to the top-level hierarchy name.
QSYS_SIMDIR	If the simulation files generated by Platform Designer are not in the simulation working directory, use the QSYS_SIMDIR variable to specify the directory location of the Platform Designer simulation files.
QUARTUS_INSTALL_DIR	Points to the Quartus installation directory that contains the device family library.

#### Example 78. Top-Level Simulation HDL File for a Testbench System

The example below shows the `pattern_generator_tb` generated for a Platform Designer system called `pattern_generator`. The `top.sv` file defines the top-level module that instantiates the `pattern_generator_tb` simulation model, as well as a custom SystemVerilog test program with BFM transactions, called `test_program`.

```
module top();
    pattern_generator_tb tb();
    test_program pgm();
endmodule
```



**Note:** The VHDL version of the Altera Tristate Conduit BFM is not supported in Synopsys VCS, NCSim, and Riviera-PRO in the Intel Quartus Prime software version 14.0. These simulators do not support the VHDL protected type, which is used to implement the BFM. For a workaround, use a simulator that supports the VHDL protected type.

**Note:** ModelSim - Intel FPGA Edition now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation. Therefore, Intel simulation libraries may not be compatible with single language simulators. If you have a VHDL-only license, some versions of Mentor simulators may not be able to simulate IP written in Verilog. As a workaround, you can use ModelSim - Intel FPGA Edition, or purchase a mixed language simulation license from Mentor.

### Related Links

[Incorporating IP Simulation Scripts in Top-Level Scripts](#)

#### 9.14.4.1 Generating a Combined Simulator Setup Script

Run the **Generate Simulator Setup Script for IP** command to generate a combined simulator setup script.

Source this combined script from a top-level simulation script. Click **Tools > Generate Simulator Setup Script for IP** (or use of the `ip-setup-simulation` utility at the command-line) to generate or update the combined scripts, after any of the following occur:

- IP core initial generation or regeneration with new parameters
- Intel Quartus Prime software version upgrade
- IP core version upgrade

To generate a combined simulator setup script for all project IP cores for each simulator:

1. Generate, regenerate, or upgrade one or more IP core. Refer to *Generating IP Cores* or *Upgrading IP Cores*.
2. Click **Tools > Generate Simulator Setup Script for IP** (or run the `ip-setup-simulation` utility). Specify the **Output Directory** and library compilation options. Click **OK** to generate the file. By default, the files generate into the `/ <project directory>/<simulator>/` directory using relative paths.
3. To incorporate the generated simulator setup script into your top-level simulation script, refer to the template section in the generated simulator setup script as a guide to creating a top-level script:
  - a. Copy the specified template sections from the simulator-specific generated scripts and paste them into a new top-level file.
  - b. Remove the comments at the beginning of each line from the copied template sections.
  - c. Specify the customizations you require to match your design simulation requirements, for example:



- Specify the `TOP_LEVEL_NAME` variable to the design’s simulation top-level file. The top-level entity of your simulation is often a testbench that instantiates your design. Then, your design instantiates IP cores or Platform Designer systems. Set the value of `TOP_LEVEL_NAME` to the top-level entity.
  - If necessary, set the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files.
  - Compile the top-level HDL file (for example, a test program) and all other files in the design.
  - Specify any other changes, such as using the `grep` command-line utility to search a transcript file for error signatures, or e-mail a report.
4. Re-run **Tools > Generate Simulator Setup Script for IP** (or `ip-setup-simulation`) after regeneration of an IP variation.

**Table 107. Simulation Script Utilities**

Utility	Syntax
<p><code>ip-setup-simulation</code> generates a combined, version-independent simulation script for all Intel FPGA IP cores in your project. The command also automates regeneration of the script after upgrading software or IP versions. Use the <code>compile-to-work</code> option to compile all simulation files into a single work library if your simulation environment requires. Use the <code>--use-relative-paths</code> option to use relative paths whenever possible.</p>	<pre>ip-setup-simulation --quartus-project=&lt;my proj&gt; --output-directory=&lt;my_dir&gt; --use-relative-paths --compile-to-work</pre> <p><code>--use-relative-paths</code> and <code>--compile-to-work</code> are optional. For command-line help listing all options for these executables, type: <code>&lt;utility name&gt; --help</code>.</p>
<p><code>ip-make-simscript</code> generates a combined simulation script for all IP cores that you specify on the command line. Specify one or more <code>.spd</code> files and an output directory in the command. Running the script compiles IP simulation models into various simulation libraries.</p>	<pre>ip-make-simscript --spd=&lt;ipA.spd, ipB.spd&gt; --output-directory=&lt;directory&gt;</pre>

The following sections provide step-by-step instructions for sourcing each simulator setup script in your top-level simulation script.

### 9.14.5 Simulating Software Running on a Nios II Processor

To simulate the software in a system driven by a Nios II processor, generate the simulation model for the Platform Designer testbench system with the following steps:

1. In the **Generation** dialog box (**Generate > Generate Testbench System**), select **Simple, BFM for clocks and resets**.
2. For the **Create testbench simulation model** option select **Verilog** or **VHDL**.
3. Click **Generate**.
4. Open the **Nios II Software Build Tools for Eclipse**.
5. Set up an application project and board support package (BSP) for the `<system>.sopcinfo` file.
6. To simulate, right-click the application project in Eclipse, and then click **Run as > Nios II ModelSim**.

Sets up the ModelSim simulation environment, and compiles and loads the Nios II software simulation.

7. To run the simulation in ModelSim, type `run -all` in the ModelSim transcript window.
8. Set the ModelSim settings and select the Platform Designer Testbench Simulation Package Descriptor (`.spd`) file, `< system >_tb.spd`. The `.spd` file is generated with the testbench simulation model for Nios II designs and specifies the files required for Nios II simulation.

**Related Links**

- [Getting Started with the Graphical User Interface](#)  
In *Nios II Gen2 Software Developer's Handbook*
- [Getting Started from the Command Line](#)  
In *Nios II Gen2 Software Developer's Handbook*

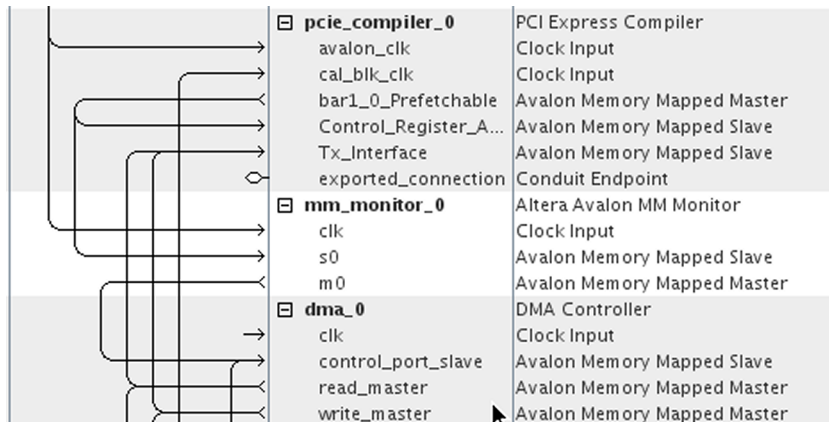
**9.14.6 Add Assertion Monitors for Simulation**

You can add monitors to Avalon-MM, AXI, and Avalon-ST interfaces in your system to verify protocol and test coverage with a simulator that supports SystemVerilog assertions.

*Note:* ModelSim - Intel FPGA Edition does not support SystemVerilog assertions. If you want to use assertion monitors, you must use a supported third-party simulators such as Mentor Questasim, Synopsys VCS, or Cadence Incisive. For more information, refer to *Introduction to Intel FPGA IP Cores*.

**Figure 186. Inserting an Avalon-MM Monitor Between an Avalon-MM Master and Slave Interface**

This example demonstrates the use of a monitor with an Avalon-MM monitor between the `pcie_compiler_bar1_0_Prefetchable` Avalon-MM master interface, and the `dma_0_control_port_slave` Avalon-MM slave interface.



Similarly, you can insert an Avalon-ST monitor between Avalon-ST source and sink interfaces.

**Related Links**

[Introduction to Intel FPGA IP Cores](#)



### 9.14.7 CMSIS Support for the HPS IP Component

Platform Designer systems that contain an HPS IP component generate a System View Description (.svd) file that lists peripherals connected to the ARM processor.

The .svd (or CMSIS-SVD) file format is an XML schema specified as part of the Cortex Microcontroller Software Interface Standard (CMSIS) provided by ARM. The .svd file allows HPS system debug tools (such as the DS-5 Debugger) to view the register maps of peripherals connected to HPS in a Platform Designer system.

#### Related Links

- [Component Interface Tcl Reference](#) on page 791
- [CMSIS - Cortex Microcontroller Software](#)

### 9.14.8 Generate Header Files

You can use the `sopc-create-header-files` command from the Nios II command shell to create header files for any master component in your Platform Designer system. The Nios II tool chain uses this command to create the processor's `system.h` file. You can also use this command to generate system level information for a hard processing system (HPS) in Intel's SoC devices or other external processors. The header file includes address map information for each slave, relative to each master that accesses the slave. Different masters may have different address maps to access a particular slave component. By default, the header files are in C format and have a `.h` suffix. You can select other formats with appropriate command-line options.

**Table 108. `sopc-create-header-files` Command-Line Options**

Option	Description
<code>&lt;sopc&gt;</code>	Path to Platform Designer <code>.sopcinfo</code> file, or the file directory. If you omit this option, the path defaults to the current directory. If you specify a directory path, you must make sure that there is a <code>.sopcinfo</code> file in the directory.
<code>--separate-masters</code>	Does not combine a module's masters that are in the same address space.
<code>--output-dir[=&lt;dirname&gt;]</code>	Allows you to specify multiple header files in <code>dirname</code> . The default output directory is <code>'.'</code>
<code>--single[=&lt;filename&gt;]</code>	Allows you to create a single header file, <code>filename</code> .
<code>--single-prefix[=&lt;prefix&gt;]</code>	Prefixes macros from a selected single master.
<code>--module[=&lt;moduleName&gt;]</code>	Specifies the module name when creating a single header file.
<code>--master[=&lt;masterName&gt;]</code>	Specifies the master name when creating a single header file.
<code>--format[=&lt;type&gt;]</code>	Specifies the header file format. Default file format is <code>.h</code> .
<code>--silent</code>	Does not display normal messages.
<code>--help</code>	Displays help for <code>sopc-create-header-files</code> .

By default, the `sopc-create-header-files` command creates multiple header files. There is one header file for the entire system, and one header file for each master group in each module. A master group is a set of masters in a module in the same address space. In general, a module may have multiple master groups. Addresses and available devices are a function of the master group.

Alternatively, you can use the `--single` option to create one header file for one master group. If there is one CPU module in the Platform Designer system with one master group, the command generates a header file for that CPU's master group. If there are no CPU modules, but there is one module with one master group, the command generates the header file for that module's master group.

You can use the `--module` and `--master` options to override these defaults. If your module has multiple master groups, use the `--master` option to specify the name of a master in the desired master group.

**Table 109. Supported Header File Formats**

Type	Suffix	Uses	Example
h	.h	C/C++ header files	<pre>#define FOO 12</pre>
m4	.m4	Macro files for m4	<pre>m4_define("FOO", 12)</pre>
sh	.sh	Shell scripts	<pre>FOO=12</pre>
mk	.mk	Makefiles	<pre>FOO := 12</pre>
pm	.pm	Perl scripts	<pre>\$macros{FOO} = 12;</pre>

**Note:** You can use the `sopc-create-header-files` command when you want to generate C macro files for DMAs that have access to memory that the Nios II does not have access to.

### 9.14.9 Incrementally Generate the System

You can modify the parameters of an IP component and regenerate the RTL for just that particular IP component.

The example below demonstrates the incremental generation flow of a Platform Designer System:

1. In Platform Designer, click **File > New System**. The **Create New System** dialog box appears, from which you create your new Platform Designer system and associate your system with a specific Intel Quartus Prime project.
2. In the IP Catalog search box, locate the **On-Chip Memory (RAM or ROM)** and double-click to add the component to your system.
3. Similarly, locate the **Reset Bridge** and **Clock Bridge** components and double-click to add the components to your system.
4. Make the necessary system connections between the IP components added to the system.

*Note:* For more information about connecting IP components, refer to *Connecting IP Components*.

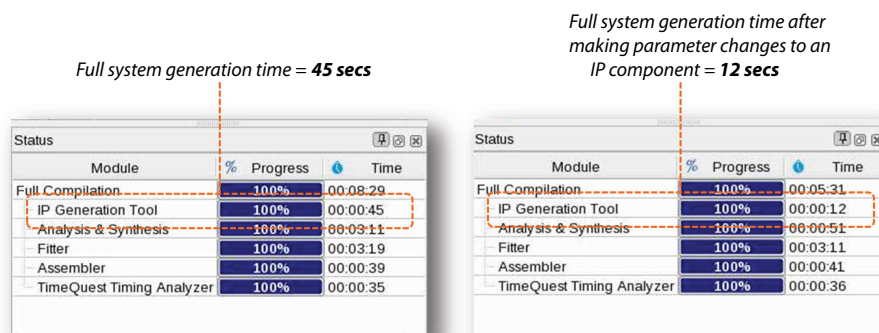
5. To save and close the system without generating, click **File > Save**.
6. In the Intel Quartus Prime software, click **File > Open Project**.
7. Select the Intel Quartus Prime project associated with your saved Platform Designer system. The Intel Quartus Prime software opens the project and the associated Platform Designer system.



8. To start the compilation of the Intel Quartus Prime project, click **Processing > Start Compilation**.
9. To open the Status window, click **View > Status**. From this window, track the time for Full Compilation, as well as IP components Generation.
10. Once the compilation finishes, in Platform Designer, click **File > Open**.
11. Select the .ip file for any one of the IP components in your saved system.
12. Modify some parameter in this .ip file.
 

*Note:* Make sure your modifications do not affect the parent system, requiring a system update by running **Validate System Integrity** from within the Platform Designer system after loading the parent system, or by running `qsys-validate` from the command-line.
13. To save the IP file, click **File > Save**.
14. To restart the compilation of the same Intel Quartus Prime project with modified Platform Designer system, click **Processing > Start Compilation** in the Intel Quartus Prime software. Platform Designer generates the RTL only for the modified IP component, skipping the generation of the other components in the system.

**Figure 187. Incremental Generation of Platform Designer System**



**Related Links**

[Connect IP Components in Your Platform Designer System](#) on page 339

**9.15 Explore and Manage Platform Designer Interconnect**

The System with Platform Designer Interconnect window allows you to see the contents of the Platform Designer interconnect before you generate your system. In this display of your system, you can review a graphical representation of the generated interconnect. Platform Designer converts connections between interfaces to interconnect logic during system generation.

You access the System with Platform Designer Interconnect window by clicking **Show System With Platform Designer Interconnect** command on the **System** menu.

The System with Platform Designer Interconnect window has the following tabs:

- **System Contents**—Displays the original instances in your system, as well as the inserted interconnect instances. Connections between interfaces are replaced by connections to interconnect where applicable.
- **Hierarchy**—Displays a system hierarchical navigator, expanding the system contents to show modules, interfaces, signals, contents of subsystems, and connections.
- **Parameters**—Displays the parameters for the selected element in the **Hierarchy** tab.
- **Memory-Mapped Interconnect**—Allows you to select a memory-mapped interconnect module and view its internal command and response networks. You can also insert pipeline stages to achieve timing closure.

The **System Contents**, **Hierarchy**, and **Parameters** tabs are read-only. Edits that you apply on the **Memory-Mapped Interconnect** tab are automatically reflected on the **Interconnect Requirements** tab.

The **Memory-Mapped Interconnect** tab in the System with Platform Designer Interconnect window displays a graphical representation of command and response datapaths in your system. Datapaths allow you precise control over pipelining in the interconnect. Platform Designer displays separate figures for the command and response datapaths. You can access the datapaths by clicking their respective tabs in the **Memory-Mapped Interconnect** tab.

Each node element in a figure represents either a master or slave that communicates over the interconnect, or an interconnect sub-module. Each edge is an abstraction of connectivity between elements, and its direction represents the flow of the commands or responses.

Click **Highlight Mode (Path, Successors, Predecessors)** to identify edges and datapaths between modules. Turn on **Show Pipelinable Locations** to add greyed-out registers on edges where pipelining is allowed in the interconnect.

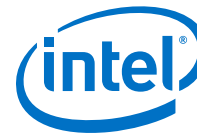
*Note:* You must select more than one module to highlight a path.

### 9.15.1 Manually Controlling Pipelining in the Platform Designer Interconnect

The **Memory-Mapped Interconnect** tab allows you to manipulate pipeline connections in the Platform Designer interconnect. Access the **Memory-Mapped Interconnect** tab by clicking **System > Show System With Platform Designer Interconnect**

*Note:* To increase interconnect frequency, you should first try increasing the value of the **Limit interconnect pipeline stages to** option on the **Interconnect Requirements** tab. You should only consider manually pipelining the interconnect if changes to this option do not improve frequency, and you have tried all other options to achieve timing closure, including the use of a bridge. Manually pipelining the interconnect should only be applied to complete systems.





1. In the **Interconnect Requirements** tab, first try increasing the value of the **Limit interconnect pipeline stages to** option until it no longer gives significant improvements in frequency, or until it causes unacceptable effects on other parts of the system.
2. In the Intel Quartus Prime software, compile your design and run timing analysis.
3. Using the timing report, identify the critical path through the interconnect and determine the approximate mid-point. The following is an example of a timing report:

```
2.800 0.000 cpu_instruction_master|out_shifter[63]|q
3.004 0.204 mm_domain_0|addr_router_001|Equal5~0|datac
3.246 0.242 mm_domain_0|addr_router_001|Equal5~0|combout
3.346 0.100 mm_domain_0|addr_router_001|Equal5~1|dataa
3.685 0.339 mm_domain_0|addr_router_001|Equal5~1|combout
4.153 0.468 mm_domain_0|addr_router_001|src_channel[5]~0|datad
4.373 0.220 mm_domain_0|addr_router_001|src_channel[5]~0|combout
```

4. In Platform Designer, click **System > Show System With Platform Designer Interconnect**.
5. In the **Memory-Mapped Interconnect** tab, select the interconnect module that contains the critical path. You can determine the name of the module from the hierarchical node names in the timing report.
6. Click **Show Pipelinable Locations**. Platform Designer display all possible pipeline locations in the interconnect. Right-click the possible pipeline location to insert or remove a pipeline stage.
7. Locate the possible pipeline location that is closest to the mid-point of the critical path. The names of the blocks in the memory-mapped interconnect tab correspond to the module instance names in the timing report.
8. Right-click the location where you want to insert a pipeline, and then click **Insert Pipeline**.
9. Regenerate the Platform Designer system, recompile the design, and then rerun timing analysis. If necessary, repeat the manual pipelining process again until timing requirements are met.

Manual pipelining has the following limitations:

- If you make changes to your original system's connectivity after manually pipelining an interconnect, your inserted pipelines may become invalid. Platform Designer displays warning messages when you generate your system if invalid pipeline stages are detected. You can remove invalid pipeline stages with the **Remove Stale Pipelines** option in the **Memory-Mapped Interconnect** tab. Intel recommends that you do not make changes to the system's connectivity after manual pipeline insertion.
- Review manually-inserted pipelines when upgrading to newer versions of Platform Designer. Manually-inserted pipelines in one version of Platform Designer may not be valid in a future version.

### Related Links

[Specify Platform Designer Interconnect Requirements](#) on page 371



## 9.16 Implement Performance Monitoring

Use the Platform Designer **Instrumentation** tab (**View > Instrumentation**) in to set up real-time performance monitoring using throughput metrics such as read and write transfers. The **Add debug instrumentation to the Platform Designer Interconnect** option allows you to interact with the Bus Analyzer Toolkit, which you can access on the **Tools** menu in the Intel Quartus Prime software.

Platform Designer supports performance monitoring for only Avalon-MM interfaces. In your Platform Designer system, you can monitor the performance of no less than three, and no greater than 15 components at one time. The performance monitoring feature works with Intel Quartus Prime software devices 13.1 and newer.

*Note:* For more information about the Bus Analyzer Toolkit and the Platform Designer Instrumentation tab, refer to the **Bus Analyzer Toolkit** page.

### Related Links

[Bus Analyzer Toolkit](#)

## 9.17 Platform Designer 64-Bit Addressing Support

Platform Designer interconnect supports up to 64-bit addressing for all Platform Designer interfaces and IP components, with a range of: 0x0000 0000 0000 0000 to 0xFFFF FFFF FFFF FFFF, inclusive.

Address parameters appear in the **Base** and **End** columns in the **System Contents** tab, on the **Address Map** tab, in the parameter editor, and in validation messages. Platform Designer displays as many digits as needed in order to display the top-most set bit, for example, 12 hex digits for a 48-bit address.

A Platform Designer system can have multiple 64-bit masters, with each master having its own address space. You can share slaves between masters and masters can map slaves to different addresses. For example, one master can interact with slave 0 at base address 0000\_0000\_0000, and another master can see the same slave at base address c000\_000\_000.

Intel Quartus Prime debugging tools provide access to the state of an addressable system via the Avalon-MM interconnect. These are also 64-bit compatible, and process within a 64-bit address space, including a JTAG to Avalon master bridge.

### Related Links

[Address Span Extender](#) on page 932

### 9.17.1 Support for Avalon-MM Non-Power of Two Data Widths

Platform Designer requires that you connect all multi-point Avalon-MM connections to interfaces with data widths that are equal to powers of two.

Platform Designer issues a validation error if an Avalon-MM master or slave interface on a multi-point connection is parameterized with a non-power of two data width.

*Note:* Avalon-MM point-to-point connections between an Avalon-MM master and an Avalon-MM slave are an exception and may set their data widths to a non-power of two.



## 9.18 Platform Designer System Example Designs

Click the **Example Design** button in the parameter editor to generate an example design.

If there are multiple example designs for an IP component, then there is a button for each example in the parameter editor. When you click the **Example Design** button, the **Select Example Design Directory** dialog box appears, where you can select the directory to save the example design.

The **Example Design** button does not appear in the parameter editor if there is no example. For some IP components, you can click **Generate > Generate Example Design** to access an example design.

The following Platform Designer system example designs demonstrate various design features and flows that you can replicate in your Platform Designer system.

### Related Links

- [Nios II Platform Designer Example Design](#)
- [PCI Express Avalon-ST Platform Designer Example Design](#)
- [Triple Speed Ethernet Platform Designer Example Design](#)

## 9.19 Platform Designer Command-Line Utilities

You can perform many of the functions available in the Platform Designer GUI at the command-line, with Platform Designer command-line utilities.

You run Platform Designer command-line executables from the Intel Quartus Prime installation directory:

```
<Intel Quartus Prime installation directory>\quartus\sopc_builder  
\bin
```

For command-line help listing of all the options for any executable, type the following command:

```
<Intel Quartus Prime installation directory>\quartus\sopc_builder  
\bin\<executable name> --help
```

**Note:** You must add `$QUARTUS_ROOTDIR/sopc_builder/bin/` to the `PATH` variable to access command-line utilities. Once you add this `PATH` variable, you can launch the utility from any directory location.

### 9.19.1 Run the Platform Designer Editor with `qsys-edit`

You can use the `qsys-edit` utility to run the Platform Designer editor from command-line.

You can use the following options with the `qsys-edit` utility:

**Table 110. qsys-edit Command-Line Options**

Option	Usage	Description
<i>1st arg file</i>	Optional	Specifies the name of the <code>.qsys</code> system or <code>.qvar</code> variation file to edit.
<code>--search-path[=&lt;value&gt;]</code>	Optional	If you omit this command, Platform Designer uses a standard default path. If you provide a search path, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example: <code>/extra/dir,\$</code> .
<code>--quartus-project[=&lt;value&gt;]</code>	Required	This option is mandatory if you are associating your Platform Designer system with an existing Intel Quartus Prime project. Specifies the name of the Intel Quartus Prime project file. If you do not provide the revision via <code>--rev</code> , Platform Designer uses the default revision as the Intel Quartus Prime project name.
<code>--new-quartus-project[=&lt;value&gt;]</code>	Required	This option is mandatory if you are associating your Platform Designer system with a new Intel Quartus Prime project. Specifies the name and path of the new Intel Quartus Prime project. Creates a new Intel Quartus Prime project at the specified path. You can also provide the revision name.
<code>--rev[=&lt;value&gt;]</code>	Optional	Specifies the name of the Intel Quartus Prime project revision.
<code>--family[=&lt;value&gt;]</code>	Optional	Sets the device family.
<code>--part[=&lt;value&gt;]</code>	Optional	Sets the device part number. If set, this option overrides the <code>--family</code> option.
<code>--new-component-type[=&lt;value&gt;]</code>	Optional	Specifies the instance type for parameterization in a variation.
<code>--require-generation</code>	Optional	Marks the loading system as requiring generation.
<code>--debug</code>	Optional	Enables debugging features and output.
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	The maximum memory size that Platform Designer uses when running <code>qsys-edit</code> . You specify this value as <code>&lt;size&gt;&lt;unit&gt;</code> , where unit is <code>m</code> (or <code>M</code> ) for multiples of megabytes, or <code>g</code> (or <code>G</code> ) for multiples of gigabytes. The default value is 512m.
<code>--help</code>	Optional	Displays help for <code>qsys-edit</code> .

### 9.19.2 Scripting IP Core Generation

Use the `qsys-script` and `qsys-generate` utilities to define and generate an IP core variation outside of the Intel Quartus Prime GUI.

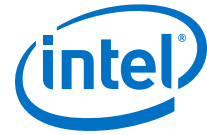
To parameterize and generate an IP core at command-line, follow these steps:

1. Run `qsys-script` to start a Tcl script that instantiates the IP and sets desired parameters:

```
qsys-script --script=<script_file>.tcl
```

2. Run `qsys-generate` to generate the IP core variation:

```
qsys-generate <IP variation file>.qsys
```



**Table 111. qsys-generate Command-Line Options**

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the .qsys system file to generate.
--synthesis=<VERILOG VHDL>	Optional	Creates synthesis HDL files that Platform Designer uses to compile the system in an Intel Quartus Prime project. Specify the generation language for the top-level RTL file for the Platform Designer system. The default value is <i>VERILOG</i> .
--block-symbol-file	Optional	Creates a Block Symbol File (.bsf) for the Platform Designer system.
--greybox	Optional	If you are synthesizing your design with a third-party EDA synthesis tool, generate a netlist for the synthesis tool to estimate timing and resource usage for this design.
--ipxact	Optional	If you set this option to true, Platform Designer renders the post-generation system as an IPXACT-compatible component description.
--simulation=<VERILOG VHDL>	Optional	Creates a simulation model for the Platform Designer system. The simulation model contains generated HDL files for the simulator, and may include simulation-only features. Specify the preferred simulation language. The default value is <i>VERILOG</i> .
--testbench=<SIMPLE STANDARD>	Optional	Creates a testbench system that instantiates the original system, adding bus functional models (BFMs) to drive the top-level interfaces. When you generate the system, the BFMs interact with the system in the simulator. The default value is <i>STANDARD</i> .
--testbench-simulation=<VERILOG VHDL>	Optional	After you create the testbench system, create a simulation model for the testbench system. The default value is <i>VERILOG</i> .
--example-design=<value>	Optional	Creates example design files. For example, --example-design or --example-design=all. The default is <i>All</i> , which generates example designs for all instances. Alternatively, choose specific filesets based on instance name and fileset name. For example --example-design=instance0.example_design1,instance1.example_design 2. Specify an output directory for the example design files creation.
--search-path=<value>	Optional	If you omit this command, Platform Designer uses a standard default path. If you provide this command, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example, "/extra/dir,\$".
--family=<value>	Optional	Sets the device family name.
--part=<value>	Optional	Sets the device part number. If set, this option overrides the --family option.
--upgrade-variation-file	Optional	If you set this option to true, the file argument for this command accepts a .v file, which contains a IP variant. This file parameterizes a corresponding instance in a Platform Designer system of the same name.
--upgrade-ip-cores	Optional	Enables upgrading all the IP cores that support upgrade in the Platform Designer system.

*continued...*

Option	Usage	Description
<code>--clear-output-directory</code>	Optional	Clears the output directory corresponding to the selected target, that is, simulation or synthesis.
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	The maximum memory size that Platform Designer uses when running <code>qsys-generate</code> . You specify the value as <code>&lt;size&gt;&lt;unit&gt;</code> , where <code>unit</code> is <code>m</code> (or <code>M</code> ) for multiples of megabytes or <code>g</code> (or <code>G</code> ) for multiples of gigabytes. The default value is <code>512m</code> .
<code>--help</code>	Optional	Displays help for <code>--qsys-generate</code> .

### 9.19.3 Display Available IP Components with ip-catalog

The `ip-catalog` command displays a list of available IP components relative to the current Intel Quartus Prime project directory, as either text or XML.

You can use the following options with the `ip-catalog` utility:

**Table 112. ip-catalog Command-Line Options**

Option	Usage	Description
<code>--project-dir= &lt;directory&gt;</code>	Optional	Finds IP components relative to the Intel Quartus Prime project directory. By default, Platform Designer uses <code>.</code> as the current directory. To exclude a project directory, leave the value empty.
<code>--type</code>	Optional	Provides a pattern to filter the type of available plug-ins. By default, Platform Designer shows only IP components. To look for a partial type string, surround with <code>*</code> , for instance, <code>*connection*</code> .
<code>--name=&lt;value&gt;</code>	Optional	Provides a pattern to filter the names of the IP components found. To show all IP components, use a <code>*</code> or <code>``</code> . By default, Platform Designer shows all IP components. The argument is not case sensitive. To look for a partial name, surround with <code>*</code> , for instance, <code>*uart*</code>
<code>--verbose</code>	Optional	Reports the progress of the command.
<code>--xml</code>	Optional	Generates the output in XML format, in place of colon-delimited format.
<code>--search-path=&lt;value&gt;</code>	Optional	If you omit this command, Platform Designer uses a standard default path. If you provide this command, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use <code>"\$"</code> , for example, <code>"/extra/dir,\$"</code> .
<code>&lt;1st arg value&gt;</code>	Optional	Specifies the directory or name fragment.
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	The maximum memory size that Platform Designer uses for when running <code>ip-catalog</code> . You specify the value as <code>&lt;size&gt;&lt;unit&gt;</code> , where <code>unit</code> is <code>m</code> (or <code>M</code> ) for multiples of megabytes or <code>g</code> (or <code>G</code> ) for multiples of gigabytes. The default value is <code>512m</code> .
<code>--help</code>	Optional	Displays help for the <code>ip-catalog</code> command.

### 9.19.4 Create an .ipx File with ip-make-ipx

The `ip-make-ipx` command creates an **.ipx** index file. This file provides a convenient way to include a collection of IP components from an arbitrary directory. You can edit the **.ipx** file to disable visibility of one or more IP components in the IP Catalog.



You can use the following options with the `ip-make-ipx` utility:

**Table 113. ip-make-ipx Command-Line Options**

Option	Usage	Description
<code>--source-directory=&lt;directory&gt;</code>	Optional	Specifies the directory containing your IP components. The default directory is <code>`.`</code> . You can provide a comma-separated list of directories.
<code>--output=&lt;file&gt;</code>	Optional	Specifies the name of the index file to generate. The default name is <code>/component.ipx</code> . Set as <code>--output=&lt;" "&gt;</code> to print the output to the console.
<code>--relative-vars=&lt;value&gt;</code>	Optional	Causes the output file to include references relative to the specified variable(s) wherever possible. You can specify multiple variables as a comma-separated list.
<code>--thorough-descent</code>	Optional	If you set this option, Platform Designer searches all the component files, without skipping the sub-directories.
<code>--message-before=&lt;value&gt;</code>	Optional	Prints a log message at the start of reading an index file.
<code>--message-after=&lt;value&gt;</code>	Optional	Prints a log message at the end of reading an index file.
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	The maximum memory size Platform Designer uses when running <code>ipr-make-ipx</code> . You specify this value as <code>&lt;size&gt;&lt;unit&gt;</code> , where unit is <code>m</code> (or <code>M</code> ) for multiples of megabytes, or <code>g</code> (or <code>G</code> ) for multiples of gigabytes. The default value is <code>512m</code> .
<code>--help</code>	Optional	Displays help for the <code>ip-make-ipx</code> command.

**Related Links**

[Set up the IP Index File \(.ipx\) to Search for IP Components](#) on page 332

**9.19.5 Generate Simulation Scripts**

You can use the `ip-make-simscript` utility to generate simulation scripts for one or more simulators, given one or more **Simulation Package Descriptor** file(s).

You can use the following options with the `ip-make-simscript` utility:

**Table 114. ip-make-simscript Command-Line Options**

Option	Usage	Description
<code>--spd[=&lt;file&gt;]</code>	Required/Repeatable	The SPD files describe the list of files that require compilation, and memory models hierarchy. This argument can either be a single path to an SPD file or a comma-separated list of paths of SPD files. For instance, <code>--spd=ipcore_1.spd,ipcore_2.spd</code>
<code>--output-directory[=&lt;directory&gt;]</code>	Optional	Specifies the directory path for the location of output files. If you do not specify a directory, the output directory defaults to the directory from which <code>--ip-make-simscript</code> runs.
<code>--compile-to-work</code>	Optional	Compiles all design files to the default library - <code>work</code> .
<code>--use-relative-paths</code>	Optional	Uses relative paths whenever possible.
<code>--cache-file[=&lt;file&gt;]</code>	Optional	Generates cache file for managed flow.
<i>continued...</i>		



Option	Usage	Description
--quiet	Optional	Quiet reporting mode. Does not report generated files.
--jvm-max-heap-size=<value>	Optional	The maximum memory size Platform Designer uses when running ip-make-simscript. You specify this value as <size><unit>, where unit is m (or M) for multiples of megabytes, or g (or G) for multiples of gigabytes. The default value is 512m.
--help	Optional	Displays help for --ip-make-simscript.

### 9.19.6 Generate a Platform Designer System with qsys-script

You can use the qsys-script utility to create and manipulate a Platform Designer system with Tcl scripting commands. If you specify a system, Platform Designer loads that system before executing any of the scripting commands.

**Note:** You must provide a package version for the qsys-script. If you do not specify the --package-version=<value> command, you must then provide a Tcl script and request the system scripting API directly with the package require -exact qsys < version > command.

#### Example 79. Platform Designer Command-Line Scripting Example

```
qsys-script --script=my_script.tcl \
--system-file=fancy.qsys my_script.tcl contains:
package require -exact qsys 16.0
# get all instance names in the system and print one by one
set instances [ get_instances ]
foreach instance $instances {
    send_message Info "$instance"
}
```

You can use the following options with the qsys-script utility:

**Table 115. qsys-script Command-Line Options**

Option	Usage	Description
--system-file=<file>	Optional	Specifies the path to a .qsys file. Platform Designer loads the system before running scripting commands.
--script=<file>	Optional	A file that contains Tcl scripting commands that you can use to create or manipulate a Platform Designer system. If you specify both --cmd and --script, Platform Designer runs the --cmd commands before the script specified by --script.
--cmd=<value>	Optional	A string that contains Tcl scripting commands that you can use to create or manipulate a Platform Designer system. If you specify both --cmd and --script, Platform Designer runs the --cmd commands before the script specified by --script.
--package-version=<value>	Optional	Specifies which Tcl API scripting version to use and determines the functionality and behavior of the Tcl commands. The Intel Quartus Prime software supports Tcl API scripting commands. The minimum supported version is 12.0. If you do not specify the version on the command-line, your script must request the scripting API directly with the package require -exact qsys <version> command.

*continued...*





Option	Usage	Description
<code>--search-path=&lt;value&gt;</code>	Optional	If you omit this command, a Platform Designer uses a standard default path. If you provide this command, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example, <code>/ &lt;directory path&gt;/dir,\$</code> . Separate multiple directory references with a comma.
<code>--quartus-project=&lt;value&gt;</code>	Optional	Specifies the path to a .qpf Intel Quartus Prime project file. Utilizes the specified Intel Quartus Prime project to add the file saved using <code>save_system</code> command. If you omit this command, Platform Designer uses the default revision as the project name.
<code>--new-quartus-project=&lt;value&gt;</code>	Optional	Specifies the name of the new Intel Quartus Prime project. Creates a new Intel Quartus Prime project at the specified path and adds the file saved using <code>save_system</code> command to the project. If you omit this command, Platform Designer uses the Intel Quartus Prime project revision as the new Intel Quartus Prime project name.
<code>--rev=&lt;value&gt;</code>	Optional	Allows you to specify the name of the Intel Quartus Prime project revision.
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	The maximum memory size that the <code>qsys-script</code> tool uses. You specify this value as <code>&lt;size&gt;&lt;unit&gt;</code> , where unit is <code>m</code> (or <code>M</code> ) for multiples of megabytes, or <code>g</code> (or <code>G</code> ) for multiples of gigabytes.
<code>--help</code>	Optional	Displays help for the <code>qsys-script</code> utility.

### Related Links

[Altera Wiki Platform Designer Scripts](#)

## 9.19.7 Platform Designer Scripting Command Reference

Platform Designer system scripting provides Tcl commands to manipulate your system. The `qsys-script` provides a command-line alternative to the Platform Designer tool. Use the `qsys-script` commands to create and modify your system, as well as to create reports about the system.

To use the current version of the Tcl commands, include the following line at the top of your script:

```
package require -exact qsys <version>
```

For example, for the current release of the Intel Quartus Prime software, include:

```
package require -exact qsys 17.0
```

The Platform Designer scripting commands fall under the following categories:

- [System](#) on page 403
- [Subsystems](#) on page 416
- [Instances](#) on page 425
- [Instantiations](#) on page 458
- [Components](#) on page 497
- [Connections](#) on page 523



[Top-level Exports](#) on page 535

[Validation](#) on page 549

[Miscellaneous](#) on page 560



### 9.19.7.1 System

This section lists the commands that allow you to manipulate your Platform Designer system.

- [create\\_system](#) on page 404
- [export\\_hw\\_tcl](#) on page 405
- [get\\_device\\_families](#) on page 406
- [get\\_devices](#) on page 407
- [get\\_module\\_properties](#) on page 408
- [get\\_module\\_property](#) on page 409
- [get\\_project\\_properties](#) on page 410
- [get\\_project\\_property](#) on page 411
- [load\\_system](#) on page 412
- [save\\_system](#) on page 413
- [set\\_module\\_property](#) on page 414
- [set\\_project\\_property](#) on page 415



### 9.19.7.1.1 create\_system

#### Description

Replaces the current system with a new system of the specified name.

#### Usage

```
create_system [<name>]
```

#### Returns

No return value.

#### Arguments

*name (optional)* The new system name.

#### Example

```
create_system my_new_system_name
```

#### Related Links

- [load\\_system](#) on page 412
- [save\\_system](#) on page 413



### 9.19.7.1.2 export\_hw\_tcl

#### Description

Allows you to save the currently open system as an `_hw.tcl` file in the project directory. The saved systems appears under the **System** category in the IP Category.

#### Usage

```
export_hw_tcl
```

#### Returns

No return value.

#### Arguments

No arguments

#### Example

```
export_hw_tcl
```

#### Related Links

- [load\\_system](#) on page 412
- [save\\_system](#) on page 413



### 9.19.7.1.3 get\_device\_families

#### Description

Returns the list of installed device families.

#### Usage

```
get_device_families
```

#### Returns

*String[]* The list of device families.

#### Arguments

No arguments

#### Example

```
get_device_families
```

#### Related Links

[get\\_devices](#) on page 407



#### 9.19.7.1.4 get\_devices

##### Description

Returns the list of installed devices for the specified family.

##### Usage

`get_devices <family>`

##### Returns

*String[]* The list of devices.

##### Arguments

*family* Specifies the family name to get the devices for.

##### Example

```
get_devices exampleFamily
```

##### Related Links

[get\\_device\\_families](#) on page 406



### 9.19.7.1.5 `get_module_properties`

#### Description

Returns the properties that you can manage for a top-level module of the Platform Designer system.

#### Usage

```
get_module_properties
```

#### Returns

The list of property names.

#### Arguments

No arguments.

#### Example

```
get_module_properties
```

#### Related Links

- [get\\_module\\_property](#) on page 409
- [set\\_module\\_property](#) on page 414





### 9.19.7.1.6 get\_module\_property

#### Description

Returns the value of a top-level system property.

#### Usage

```
get_module_property <property>
```

#### Returns

The property value.

#### Arguments

*property* The property name to query. Refer to *Module Properties*.

#### Example

```
get_module_property NAME
```

#### Related Links

- [get\\_module\\_properties](#) on page 408
- [set\\_module\\_property](#) on page 414



### 9.19.7.1.7 get\_project\_properties

#### Description

Returns the list of properties that you can query for properties pertaining to the Intel Quartus Prime project.

#### Usage

```
get_project_properties
```

#### Returns

The list of project properties.

#### Arguments

No arguments

#### Example

```
get_project_properties
```

#### Related Links

- [get\\_project\\_property](#) on page 411
- [set\\_project\\_property](#) on page 415



### 9.19.7.1.8 get\_project\_property

#### Description

Returns the value of a Intel Quartus Prime project property.

#### Usage

```
get_project_property <property>
```

#### Returns

The property value.

#### Arguments

*property* The project property name. Refer to *Project properties*.

#### Example

```
get_project_property DEVICE_FAMILY
```

#### Related Links

- [get\\_module\\_properties](#) on page 408
- [get\\_module\\_property](#) on page 409
- [set\\_module\\_property](#) on page 414
- [Project Properties](#) on page 588



### 9.19.7.1.9 load\_system

#### Description

Loads the Platform Designer system from a file, and uses the system as the current system for scripting commands.

#### Usage

```
load_system <file>
```

#### Returns

No return value.

#### Arguments

*file* The path to the .qsys file.

#### Example

```
load_system example.qsys
```

#### Related Links

- [create\\_system](#) on page 404
- [save\\_system](#) on page 413



### 9.19.7.1.10 save\_system

#### Description

Saves the current system to the specified file. If you do not specify the file, Platform Designer saves the system to the same file opened with the `load_system` command.

#### Usage

```
save_system <file>
```

#### Returns

No return value.

#### Arguments

*file* If available, the path of the **.qsys** file to save.

#### Example

```
save_system
```

```
save_system file.qsys
```

#### Related Links

- [load\\_system](#) on page 412
- [create\\_system](#) on page 404



### 9.19.7.1.11 set\_module\_property

#### Description

Specifies the Tcl procedure to evaluate changes in Platform Designer system instance parameters.

#### Usage

```
set_module_property <property> <value>
```

#### Returns

No return value.

#### Arguments

*property* The property name. Refer to *Module Properties*.

*value* The new value of the property.

#### Example

```
set_module_property COMPOSITION_CALLBACK "my_composition_callback"
```

#### Related Links

- [get\\_module\\_properties](#) on page 408
- [get\\_module\\_property](#) on page 409
- [Module Properties](#) on page 582



### 9.19.7.1.12 set\_project\_property

#### Description

Sets the project property value, such as the device family.

#### Usage

```
set_project_property <property> <value>
```

#### Returns

No return value.

#### Arguments

*property* The property name. Refer to *Project Properties*.

*value* The new property value.

#### Example

```
set_project_property DEVICE_FAMILY "Cyclone IV GX"
```

#### Related Links

- [get\\_project\\_properties](#) on page 410
- [get\\_project\\_property](#) on page 411
- [Project Properties](#) on page 588



### 9.19.7.2 Subsystems

This section lists the commands that allow you to obtain the connection and parameter information of instances in your Platform Designer subsystem.

[get\\_composed\\_connections](#) on page 417

[get\\_composed\\_connection\\_parameter\\_value](#) on page 418

[get\\_composed\\_connection\\_parameters](#) on page 419

[get\\_composed\\_instance\\_assignment](#) on page 420

[get\\_composed\\_instance\\_assignments](#) on page 421

[get\\_composed\\_instance\\_parameter\\_value](#) on page 422

[get\\_composed\\_instance\\_parameters](#) on page 423

[get\\_composed\\_instances](#) on page 424





### 9.19.7.2.1 `get_composed_connections`

#### Description

Returns the list of all connections in the subsystem for an instance that contains the subsystem of the Platform Designer system.

#### Usage

```
get_composed_connections <instance>
```

#### Returns

The list of connection names in the subsystem.

#### Arguments

*instance* The child instance containing the subsystem.

#### Example

```
get_composed_connections subsystem_0
```

#### Related Links

- [get\\_composed\\_connection\\_parameter\\_value](#) on page 418
- [get\\_composed\\_connection\\_parameters](#) on page 419



### 9.19.7.2.2 `get_composed_connection_parameter_value`

#### Description

Returns the parameter value of a connection in a child instance containing the subsystem.

#### Usage

```
get_composed_connection_parameter_value <instance> <child_connection>  
<parameter>
```

#### Returns

The parameter value.

#### Arguments

*instance* The child instance that contains the subsystem.

*child\_connection* The connection name in the subsystem.

*parameter* The parameter name to query for the connection.

#### Example

```
get_composed_connection_parameter_value subsystem_0 cpu.data_master/memory.s0  
baseAddress
```

#### Related Links

- [get\\_composed\\_connection\\_parameters](#) on page 419
- [get\\_composed\\_connections](#) on page 417



### 9.19.7.2.3 `get_composed_connection_parameters`

#### Description

Returns the list of parameters of a connection in the subsystem, for an instance that contains the subsystem.

#### Usage

```
get_composed_connection_parameters <instance> <child_connection>
```

#### Returns

The list of parameter names.

#### Arguments

*instance* The child instance containing the subsystem.

*child\_connection* The name of the connection in the subsystem.

#### Example

```
get_composed_connection_parameters subsystem_0 cpu.data_master/memory.s0
```

#### Related Links

- [get\\_composed\\_connection\\_parameter\\_value](#) on page 418
- [get\\_composed\\_connections](#) on page 417



#### 9.19.7.2.4 `get_composed_instance_assignment`

##### Description

Returns the assignment value of the child instance in the subsystem.

##### Usage

```
get_composed_instance_assignment <instance> <child_instance>  
<assignment>
```

##### Returns

The assignment value.

##### Arguments

*instance* The subsystem containing the child instance.

*child\_instance* The child instance name in the subsystem.

*assignment* The assignment key.

##### Example

```
get_composed_instance_assignment subsystem_0 video_0  
"embeddedsw.CMacro.colorSpace"
```

##### Related Links

- [get\\_composed\\_instance\\_assignments](#) on page 421
- [get\\_composed\\_instances](#) on page 424



### 9.19.7.2.5 `get_composed_instance_assignments`

#### Description

Returns the list of assignments of the child instance in the subsystem.

#### Usage

```
get_composed_instance_assignments <instance> <child_instance>
```

#### Returns

The list of assignment names.

#### Arguments

*instance* The subsystem containing the child instance.

*child\_instance* The child instance name in the subsystem.

#### Example

```
get_composed_instance_assignments subsystem_0 cpu
```

#### Related Links

- [get\\_composed\\_instance\\_assignment](#) on page 420
- [get\\_composed\\_instances](#) on page 424



### 9.19.7.2.6 `get_composed_instance_parameter_value`

#### Description

Returns the parameter value of the child instance in the subsystem.

#### Usage

```
get_composed_instance_parameter_value <instance> <child_instance>  
<parameter>
```

#### Returns

The parameter value of the instance in the subsystem.

#### Arguments

*instance* The subsystem containing the child instance.

*child\_instance* The child instance name in the subsystem.

*parameter* The parameter name to query on the child instance in the subsystem.

#### Example

```
get_composed_instance_parameter_value subsystem_0 cpu DATA_WIDTH
```

#### Related Links

- [get\\_composed\\_instance\\_parameters](#) on page 423
- [get\\_composed\\_instances](#) on page 424



### 9.19.7.2.7 `get_composed_instance_parameters`

#### Description

Returns the list of parameters of the child instance in the subsystem.

#### Usage

```
get_composed_instance_parameters <instance> <child_instance>
```

#### Returns

The list of parameter names.

#### Arguments

*instance* The subsystem containing the child instance.

*child\_instance* The child instance name in the subsystem.

#### Example

```
get_composed_instance_parameters subsystem_0 cpu
```

#### Related Links

- [get\\_composed\\_instance\\_parameter\\_value](#) on page 422
- [get\\_composed\\_instances](#) on page 424



### 9.19.7.2.8 `get_composed_instances`

#### Description

Returns the list of child instances in the subsystem.

#### Usage

```
get_composed_instances <instance>
```

#### Returns

The list of instance names in the subsystem.

#### Arguments

*instance* The subsystem containing the child instance.

#### Example

```
get_composed_instances subsystem_0
```

#### Related Links

- [get\\_composed\\_instance\\_assignment](#) on page 420
- [get\\_composed\\_instance\\_assignments](#) on page 421
- [get\\_composed\\_instance\\_parameter\\_value](#) on page 422
- [get\\_composed\\_instance\\_parameters](#) on page 423





### 9.19.7.3 Instances

This section lists the commands that allow you to manipulate the instances of IP components in your Platform Designer system.

- [add\\_instance](#) on page 426
- [apply\\_instance\\_preset](#) on page 427
- [create\\_ip](#) on page 428
- [add\\_component](#) on page 429
- [duplicate\\_instance](#) on page 430
- [enable\\_instance\\_parameter\\_update\\_callback](#) on page 431
- [get\\_instance\\_assignment](#) on page 432
- [get\\_instance\\_assignments](#) on page 433
- [get\\_instance\\_documentation\\_links](#) on page 434
- [get\\_instance\\_interface\\_assignment](#) on page 435
- [get\\_instance\\_interface\\_assignments](#) on page 436
- [get\\_instance\\_interface\\_parameter\\_property](#) on page 437
- [get\\_instance\\_interface\\_parameter\\_value](#) on page 438
- [get\\_instance\\_interface\\_parameters](#) on page 439
- [get\\_instance\\_interface\\_port\\_property](#) on page 440
- [get\\_instance\\_interface\\_ports](#) on page 441
- [get\\_instance\\_interface\\_properties](#) on page 442
- [get\\_instance\\_interface\\_property](#) on page 443
- [get\\_instance\\_interfaces](#) on page 444
- [get\\_instance\\_parameter\\_property](#) on page 445
- [get\\_instance\\_parameter\\_value](#) on page 446
- [get\\_instance\\_parameter\\_values](#) on page 447
- [get\\_instance\\_parameters](#) on page 448
- [get\\_instance\\_port\\_property](#) on page 449
- [get\\_instance\\_properties](#) on page 450
- [get\\_instance\\_property](#) on page 451
- [get\\_instances](#) on page 452
- [is\\_instance\\_parameter\\_update\\_callback\\_enabled](#) on page 453
- [remove\\_instance](#) on page 454
- [set\\_instance\\_parameter\\_value](#) on page 455
- [set\\_instance\\_parameter\\_values](#) on page 456
- [set\\_instance\\_property](#) on page 457



### 9.19.7.3.1 add\_instance

#### Description

Adds an instance of a component, referred to as a *child* or *child instance*, to the system.

#### Usage

```
add_instance <name> <type> [<version>]
```

#### Returns

No return value.

#### Arguments

*name* Specifies a unique local name that you can use to manipulate the instance. Platform Designer uses this name in the generated HDL to identify the instance.

*type* Refers to a kind of instance available in the IP Catalog, for example `altera_avalon_uart`.

*version (optional)* The required version of the specified instance type. If you do not specify any instance, Platform Designer uses the latest version.

#### Example

```
add_instance uart_0 altera_avalon_uart 16.1
```

#### Related Links

- [get\\_instance\\_property](#) on page 451
- [get\\_instances](#) on page 452
- [remove\\_instance](#) on page 454
- [set\\_instance\\_parameter\\_value](#) on page 455
- [get\\_instance\\_parameter\\_value](#) on page 446



### 9.19.7.3.2 apply\_instance\_preset

#### Description

Applies the settings in a preset to the specified instance.

#### Usage

```
apply_instance_preset <preset_name>
```

#### Returns

No return value.

#### Arguments

*preset\_name* The preset name.

#### Example

```
apply_preset "Custom Debug Settings"
```

#### Related Links

[set\\_instance\\_parameter\\_value](#) on page 455



### 9.19.7.3.3 create\_ip

#### Description

Creates a new IP Variation system with the given instance.

#### Usage

```
create_ip <type> [ <instance_name> <version> ]
```

#### Returns

No return value.

#### Arguments

*type* Kind of instance available in the IP catalog, for example, altera\_avalon\_uart.

*instance\_name (optional)* A unique local name that you can use to manipulate the instance. If not specified, Platform Designer uses a default name.

*version (optional)* The required version of the specified instance type. If not specified, Platform Designer uses the latest version.

#### Example

```
create_ip altera_avalon_uart altera_avalon_uart_inst 17.0
```

#### Related Links

- [add\\_component](#) on page 429
- [load\\_system](#) on page 412
- [save\\_system](#) on page 413
- [set\\_instance\\_parameter\\_value](#) on page 455



### 9.19.7.3.4 add\_component

#### Description

Adds a new IP Variation component to the system.

#### Usage

```
add_component <instance_name> <file_name> [<component_type>  
<component_instance_name> <component_version>]
```

#### Returns

No return value.

#### Arguments

*instance\_name* A unique local name that you can use to manipulate the instance.

*file\_name* The IP variation file name. If a path is not specified, Platform Designer saves the file in the `./ip/system/` sub-folder of your system.

*component\_type*  
(optional) The kind of instance available in the IP catalog, for example `altera_avalon_uart`.

*component\_instance\_name*  
(optional) The instance name of the component in the IP variation file. If not specified, Platform Designer uses a default name.

*component\_version*  
(optional) The required version of the specified instance type. If not specified, Platform Designer uses the latest version.

#### Example

```
add_component myuart_0 myuart.ip altera_avalon_uart altera_avalon_uart_inst  
17.0
```

#### Related Links

- [load\\_component](#) on page 518
- [load\\_instantiation](#) on page 485
- [save\\_system](#) on page 413



### 9.19.7.3.5 duplicate\_instance

#### Description

Creates a duplicate instance of the specified instance.

#### Usage

```
duplicate_instance <instance> [ <name>]
```

#### Returns

*String* The new instance name.

#### Arguments

*instance* Specifies the instance name to duplicate.

*name (optional)* Specifies the name of the duplicate instance.

#### Example

```
duplicate_instance cpu cpu_0
```

#### Related Links

- [add\\_instance](#) on page 426
- [remove\\_instance](#) on page 454



### 9.19.7.3.6 enable\_instance\_parameter\_update\_callback

#### Description

Enables the update callback for instance parameters.

#### Usage

enable\_instance\_parameter\_update\_callback [<value>]

#### Returns

No return value.

#### Arguments

*value (optional)* Specifies whether to enable/disable the instance parameters callback. Default option is "1".

#### Example

```
enabled_instance_parameter_update_callback
```

#### Related Links

- [is\\_instance\\_parameter\\_update\\_callback\\_enabled](#) on page 453
- [set\\_instance\\_parameter\\_value](#) on page 455



### 9.19.7.3.7 `get_instance_assignment`

#### Description

Returns the assignment value of a child instance. Platform Designer uses assignments to transfer information about hardware to embedded software tools and applications.

#### Usage

```
get_instance_assignment <instance> <assignment>
```

#### Returns

*String* The value of the specified assignment.

#### Arguments

*instance* The instance name.

*assignment* The assignment key to query.

#### Example

```
get_instance_assignment video_0 embeddedsw.CMacro.colorSpace
```

#### Related Links

[get\\_instance\\_assignments](#) on page 433





### 9.19.7.3.8 `get_instance_assignments`

#### Description

Returns the list of assignment keys for any defined assignments for the instance.

#### Usage

```
get_instance_assignments <instance>
```

#### Returns

*String[]* The list of assignment keys.

#### Arguments

*instance* The instance name.

#### Example

```
get_instance_assignments sdram
```

#### Related Links

[get\\_instance\\_assignment](#) on page 432



### 9.19.7.3.9 get\_instance\_documentation\_links

#### Description

Returns the list of all documentation links provided by an instance.

#### Usage

```
get_instance_documentation_links <instance>
```

#### Returns

*String[]* The list of documentation links.

#### Arguments

*instance* The instance name.

#### Example

```
get_instance_documentation_links cpu_0
```

#### Notes

The list of documentation links includes titles and URLs for the links. For instance, a component with a single data sheet link may return:

```
{Data Sheet} {http://url/to/data/sheet}
```



### 9.19.7.3.10 `get_instance_interface_assignment`

#### Description

Returns the assignment value for an interface of a child instance. Platform Designer uses assignments to transfer information about hardware to embedded software tools and applications.

#### Usage

```
get_instance_interface_assignment <instance> <interface> <assignment>
```

#### Returns

*String* The value of the specified assignment.

#### Arguments

*instance* The child instance name.

*interface* The interface name.

*assignment* The assignment key to query.

#### Example

```
get_instance_interface_assignment sdram s1 embeddedsw.configuration.isFlash
```

#### Related Links

[get\\_instance\\_interface\\_assignments](#) on page 436



### 9.19.7.3.11 `get_instance_interface_assignments`

#### Description

Returns the list of assignment keys for any assignments defined for an interface of a child instance.

#### Usage

```
get_instance_interface_assignments <instance> <interface>
```

#### Returns

*String[]* The list of assignment keys.

#### Arguments

*instance* The child instance name.

*interface* The interface name.

#### Example

```
get_instance_interface_assignments sdram s1
```

#### Related Links

[get\\_instance\\_interface\\_assignment](#) on page 435



### 9.19.7.3.12 `get_instance_interface_parameter_property`

#### Description

Returns the property value for a parameter in an interface of an instance. Parameter properties are metadata about how Platform Designer uses the parameter.

#### Usage

```
get_instance_interface_parameter_property <instance> <interface>  
<parameter> <property>
```

#### Returns

*various* The parameter property value.

#### Arguments

*instance* The child instance name.

*interface* The interface name.

*parameter* The parameter name for the interface.

*property* The property name for the parameter. Refer to *Parameter Properties*.

#### Example

```
get_instance_interface_parameter_property uart_0 s0 setupTime ENABLED
```

#### Related Links

- [get\\_instance\\_interface\\_parameters](#) on page 439
- [get\\_instance\\_interfaces](#) on page 444
- [get\\_parameter\\_properties](#) on page 566
- [Parameter Properties](#) on page 583



### 9.19.7.3.13 `get_instance_interface_parameter_value`

#### Description

Returns the parameter value of an interface in an instance.

#### Usage

```
get_instance_interface_parameter_value <instance> <interface>  
<parameter>
```

#### Returns

*various* The parameter value.

#### Arguments

*instance* The child instance name.

*interface* The interface name.

*parameter* The parameter name for the interface.

#### Example

```
get_instance_interface_parameter_value uart_0 s0 setupTime
```

#### Related Links

- [get\\_instance\\_interface\\_parameters](#) on page 439
- [get\\_instance\\_interfaces](#) on page 444



#### 9.19.7.3.14 `get_instance_interface_parameters`

##### Description

Returns the list of parameters for an interface in an instance.

##### Usage

```
get_instance_interface_parameters <instance> <interface>
```

##### Returns

*String[]* The list of parameter names for parameters in the interface.

##### Arguments

*instance* The child instance name.

*interface* The interface name.

##### Example

```
get_instance_interface_parameters uart_0 s0
```

##### Related Links

- [get\\_instance\\_interface\\_parameter\\_value](#) on page 438
- [get\\_instance\\_interfaces](#) on page 444



### 9.19.7.3.15 `get_instance_interface_port_property`

#### Description

Returns the property value of a port in the interface of a child instance.

#### Usage

```
get_instance_interface_port_property <instance> <interface> <port>  
<property>
```

#### Returns

*various* The port property value.

#### Arguments

*instance* The child instance name.

*interface* The interface name.

*port* The port name.

*property* The property name of the port. Refer to *Port Properties*.

#### Example

```
get_instance_interface_port_property uart_0 exports tx WIDTH
```

#### Related Links

- [get\\_instance\\_interface\\_ports](#) on page 441
- [get\\_port\\_properties](#) on page 544
- [Port Properties](#) on page 587





### 9.19.7.3.16 `get_instance_interface_ports`

#### Description

Returns the list of ports in an interface of an instance.

#### Usage

```
get_instance_interface_ports <instance> <interface>
```

#### Returns

*String[]* The list of port names in the interface.

#### Arguments

*instance* The instance name.

*interface* The interface name.

#### Example

```
get_instance_interface_ports uart_0 s0
```

#### Related Links

- [get\\_instance\\_interface\\_port\\_property](#) on page 440
- [get\\_instance\\_interfaces](#) on page 444



### 9.19.7.3.17 `get_instance_interface_properties`

#### Description

Returns the list of properties that you can query for an interface in an instance.

#### Usage

```
get_instance_interface_properties
```

#### Returns

*String[]* The list of property names.

#### Arguments

No arguments.

#### Example

```
get_instance_interface_properties
```

#### Related Links

- [get\\_instance\\_interface\\_property](#) on page 443
- [get\\_instance\\_interfaces](#) on page 444



### 9.19.7.3.18 `get_instance_interface_property`

#### Description

Returns the property value for an interface in a child instance.

#### Usage

```
get_instance_interface_property <instance> <interface> <property>
```

#### Returns

*String* The property value.

#### Arguments

*instance* The child instance name.

*interface* The interface name.

*property* The property name. Refer to *Element Properties*.

#### Example

```
get_instance_interface_property uart_0 s0 DESCRIPTION
```

#### Related Links

- [get\\_instance\\_interface\\_properties](#) on page 442
- [get\\_instance\\_interfaces](#) on page 444
- [Element Properties](#) on page 578



### 9.19.7.3.19 `get_instance_interfaces`

#### Description

Returns the list of interfaces in an instance.

#### Usage

```
get_instance_interfaces <instance>
```

#### Returns

*String[]* The list of interface names.

#### Arguments

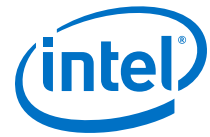
*instance* The instance name.

#### Example

```
get_instance_interfaces uart_0
```

#### Related Links

- [get\\_instance\\_interface\\_ports](#) on page 441
- [get\\_instance\\_interface\\_properties](#) on page 442
- [get\\_instance\\_interface\\_property](#) on page 443



### 9.19.7.3.20 `get_instance_parameter_property`

#### Description

Returns the property value of a parameter in an instance. Parameter properties are metadata about how Platform Designer uses the parameter.

#### Usage

```
get_instance_parameter_property <instance> <parameter> <property>
```

#### Returns

*various* The parameter property value.

#### Arguments

*instance* The instance name.

*parameter* The parameter name.

*property* The property name of the parameter. Refer to *Parameter Properties*.

#### Example

```
get_instance_parameter_property uart_0 baudRate ENABLED
```

#### Related Links

- [get\\_instance\\_parameters](#) on page 448
- [get\\_parameter\\_properties](#) on page 566
- [Parameter Properties](#) on page 583



### 9.19.7.3.21 `get_instance_parameter_value`

#### Description

Returns the parameter value in a child instance.

#### Usage

```
get_instance_parameter_value <instance> <parameter>
```

#### Returns

*various* The parameter value.

#### Arguments

*instance* The instance name.

*parameter* The parameter name.

#### Example

```
get_instance_parameter_value pixel_converter input_DPI
```

#### Related Links

- [get\\_instance\\_parameters](#) on page 448
- [set\\_instance\\_parameter\\_value](#) on page 455



### 9.19.7.3.22 `get_instance_parameter_values`

#### Description

Returns a list of the parameters' values in a child instance.

#### Usage

```
get_instance_parameter_values <instance> <parameters>
```

#### Returns

*String[]* A list of the parameters' value.

#### Arguments

*instance* The child instance name.

*parameter* A list of parameter names in the instance.

#### Example

```
get_instance_parameter_value uart_0 [list param1 param2]
```

#### Related Links

- [get\\_instance\\_parameters](#) on page 448
- [set\\_instance\\_parameter\\_value](#) on page 455
- [set\\_instance\\_parameter\\_values](#) on page 456



### 9.19.7.3.23 `get_instance_parameters`

#### Description

Returns the names of all parameters for a child instance that the parent can manipulate. This command omits derived parameters and parameters that have the `SYSTEM_INFO` parameter property set.

#### Usage

```
get_instance_parameters <instance>
```

#### Returns

*instance* The list of parameters in the instance.

#### Arguments

*instance* The instance name.

#### Example

```
get_instance_parameters uart_0
```

#### Related Links

- [get\\_instance\\_parameter\\_property](#) on page 445
- [get\\_instance\\_parameter\\_value](#) on page 446
- [set\\_instance\\_parameter\\_value](#) on page 455





### 9.19.7.3.24 `get_instance_port_property`

#### Description

Returns the property value of a port contained by an interface in a child instance.

#### Usage

```
get_instance_port_property <instance> <port> <property>
```

#### Returns

*various* The property value for the port.

#### Arguments

*instance* The child instance name.

*port* The port name.

*property* The property name. Refer to *Port Properties*.

#### Example

```
get_instance_port_property uart_0 tx WIDTH
```

#### Related Links

- [get\\_instance\\_interface\\_ports](#) on page 441
- [get\\_port\\_properties](#) on page 544
- [Port Properties](#) on page 587



### 9.19.7.3.25 `get_instance_properties`

#### Description

Returns the list of properties for a child instance.

#### Usage

```
get_instance_properties
```

#### Returns

*String[]* The list of property names for the child instance.

#### Arguments

No arguments.

#### Example

```
get_instance_properties
```

#### Related Links

[get\\_instance\\_property](#) on page 451



### 9.19.7.3.26 `get_instance_property`

#### Description

Returns the property value for a child instance.

#### Usage

```
get_instance_property <instance> <property>
```

#### Returns

*String* The property value.

#### Arguments

*instance* The child instance name.

*property* The property name. Refer to *Element Properties*.

#### Example

```
get_instance_property uart_0 ENABLED
```

#### Related Links

- [get\\_instance\\_properties](#) on page 450
- [Element Properties](#) on page 578



### 9.19.7.3.27 `get_instances`

#### Description

Returns the list of the instance names for all the instances in the system.

#### Usage

`get_instances`

#### Returns

*String[]* The list of child instance names.

#### Arguments

No arguments.

#### Example

```
get_instances
```

#### Related Links

- [add\\_instance](#) on page 426
- [remove\\_instance](#) on page 454



### 9.19.7.3.28 `is_instance_parameter_update_callback_enabled`

#### Description

Returns true if you enable the update callback for instance parameters.

#### Usage

```
is_instance_parameter_update_callback_enabled
```

#### Returns

*boolean* 1 if you enable the callback; 0 if you disable the callback.

#### Arguments

No arguments

#### Example

```
is_instance_parameter_update_callback_enabled
```

#### Related Links

[enable\\_instance\\_parameter\\_update\\_callback](#) on page 431



### 9.19.7.3.29 remove\_instance

#### Description

Removes an instance from the system.

#### Usage

```
remove_instance <instance>
```

#### Returns

No return value.

#### Arguments

*instance* The child instance name to remove.

#### Example

```
remove_instance cpu
```

#### Related Links

- [add\\_instance](#) on page 426
- [get\\_instances](#) on page 452



### 9.19.7.3.30 set\_instance\_parameter\_value

#### Description

Sets the parameter value for a child instance. You cannot set derived parameters and SYSTEM\_INFO parameters for the child instance with this command.

#### Usage

```
set_instance_parameter_value <instance> <parameter> <value>
```

#### Returns

No return value.

#### Arguments

*instance* The child instance name.

*parameter* The parameter name.

*value* The parameter value.

#### Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

#### Related Links

- [get\\_instance\\_parameter\\_value](#) on page 446
- [get\\_instance\\_parameter\\_property](#) on page 445



### 9.19.7.3.31 set\_instance\_parameter\_values

#### Description

Sets a list of parameter values for a child instance. You cannot set derived parameters and SYSTEM\_INFO parameters for the child instance with this command.

#### Usage

```
set_instance_parameter_value <instance> <parameter_value_pairs>
```

#### Returns

No return value.

#### Arguments

*instance* The child instance name.

*parameter\_value\_pairs* The pairs of parameter name and value to set.

#### Example

```
set_instance_parameter_value uart_0 [list baudRate 9600 parity odd]
```

#### Related Links

- [get\\_instance\\_parameter\\_value](#) on page 446
- [get\\_instance\\_parameter\\_values](#) on page 447
- [get\\_instance\\_parameters](#) on page 448





### 9.19.7.3.32 set\_instance\_property

#### Description

Sets the property value of a child instance. Most instance properties are read-only and can only be set by the instance itself. The primary use for this command is to update the `ENABLED` parameter, which includes or excludes a child instance when generating Platform Designer interconnect.

#### Usage

```
set_instance_property <instance> <property> <value>
```

#### Returns

No return value.

#### Arguments

*instance* The child instance name.

*property* The property name. Refer to *Instance Properties*.

*value* The property value.

#### Example

```
set_instance_property cpu ENABLED false
```

#### Related Links

- [get\\_instance\\_parameters](#) on page 448
- [get\\_instance\\_property](#) on page 451
- [Instance Properties](#) on page 579



#### 9.19.7.4 Instantiations

This section lists the commands that allow you to manipulate the loaded instantiations in your Platform Designer system.

[add\\_instantiation\\_hdl\\_file](#) on page 460  
[add\\_instantiation\\_interface](#) on page 461  
[add\\_instantiation\\_interface\\_port](#) on page 462  
[copy\\_instance\\_interface\\_to\\_instantiation](#) on page 463  
[get\\_instantiation\\_assignment\\_value](#) on page 464  
[get\\_instantiation\\_assignments](#) on page 465  
[get\\_instantiation\\_hdl\\_file\\_properties](#) on page 466  
[get\\_instantiation\\_hdl\\_file\\_property](#) on page 467  
[get\\_instantiation\\_hdl\\_files](#) on page 468  
[get\\_instantiation\\_interface\\_assignment\\_value](#) on page 469  
[get\\_instantiation\\_interface\\_assignments](#) on page 470  
[get\\_instantiation\\_interface\\_parameter\\_value](#) on page 471  
[get\\_instantiation\\_interface\\_parameters](#) on page 472  
[get\\_instantiation\\_interface\\_port\\_properties](#) on page 473  
[get\\_instantiation\\_interface\\_port\\_property](#) on page 474  
[get\\_instantiation\\_interface\\_ports](#) on page 475  
[get\\_instantiation\\_interface\\_property](#) on page 476  
[get\\_instantiation\\_interface\\_properties](#) on page 477  
[get\\_instantiation\\_interface\\_sysinfo\\_parameter\\_value](#) on page 478  
[get\\_instantiation\\_interface\\_sysinfo\\_parameters](#) on page 479  
[get\\_instantiation\\_interfaces](#) on page 480  
[get\\_instantiation\\_properties](#) on page 481  
[get\\_instantiation\\_property](#) on page 482  
[get\\_loaded\\_instantiation](#) on page 483  
[import\\_instantiation\\_interfaces](#) on page 484  
[load\\_instantiation](#) on page 485  
[remove\\_instantiation\\_hdl\\_file](#) on page 486  
[remove\\_instantiation\\_interface](#) on page 487  
[remove\\_instantiation\\_interface\\_port](#) on page 488  
[save\\_instantiation](#) on page 489  
[set\\_instantiation\\_assignment\\_value](#) on page 490  
[set\\_instantiation\\_hdl\\_file\\_property](#) on page 491  
[set\\_instantiation\\_interface\\_assignment\\_value](#) on page 492  
[set\\_instantiation\\_interface\\_parameter\\_value](#) on page 493



[set\\_instantiation\\_interface\\_port\\_property](#) on page 494

[set\\_instantiation\\_interface\\_sysinfo\\_parameter\\_value](#) on page 495

[set\\_instantiation\\_property](#) on page 496



#### 9.19.7.4.1 add\_instantiation\_hdl\_file

##### Description

Adds an HDL file to the loaded instantiation.

##### Usage

```
add_instantiation_hdl_file <file> [<kind>]
```

##### Returns

No return value.

##### Arguments

*file* Specifies the HDL file name.

*kind(optional)* Indicates the file set kind to add the file to. If you do not specify this option, the command adds the file to all the file sets. Refer to *File Set Kind*.

##### Example

```
add_instantiation_hdl_file my_nios2_gen2.vhdl quartus_synth
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [File Set Kind](#) on page 594



#### 9.19.7.4.2 add\_instantiation\_interface

##### Description

Adds an interface to the loaded instantiation.

##### Usage

```
add_instantiation_interface <interface> <type> <direction>
```

##### Returns

No return value.

##### Arguments

*interface* Specifies the interface name.

*type* Specifies the interface type.

*direction* Specifies the interface direction. Refer to *Interface Direction*.

##### Example

```
add_instantiation_interface clk_0 clock OUTPUT
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Interface Direction](#) on page 593



### 9.19.7.4.3 add\_instantiation\_interface\_port

#### Description

Adds a port to a loaded instantiation's interface.

#### Usage

```
add_instantiation_interface_port <interface> <port> <role> <width>  
<vhdl_type><direction>
```

#### Returns

No return value.

#### Arguments

*interface* Specifies the interface name.

*port* Specifies the port name.

*role* Specifies the port role.

*width* Specifies the port width.

*vhdl\_type* Specifies the VHDL type of the port. Refer to *VHDL Type*.

*direction* Specifies the port direction. Refer to *Direction Properties*.

#### Example

```
add_instantiation_interface_port avs_s0 avs_s0_address address 8 {standard  
logic vector} input
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [VHDL Type](#) on page 601
- [Direction Properties](#) on page 577



#### 9.19.7.4.4 copy\_instance\_interface\_to\_instantiation

##### Description

Adds an interface to a loaded instantiation by copying the specified interface of another instance.

##### Usage

```
copy_instance_interface_to_instantiation <instance> <interface> <type>
```

##### Returns

*String* The name of the newly added interface.

##### Arguments

*instance* Specifies the name of the instance to copy the interface from.

*interface* Specifies the name of the interface to copy.

*type* Specifies the type of copy to make. Refer to *Instantiation Interface Duplicate Type*.

##### Example

```
copy_instance_interface_to_instantiation cpu_0 data_master CLONE
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Instantiation Interface Duplicate Type](#) on page 597



#### 9.19.7.4.5 `get_instantiation_assignment_value`

##### Description

Gets the assignment value on the loaded instantiation.

##### Usage

```
get_instantiation_assignment_value <name>
```

##### Returns

*String* The assignment value.

##### Arguments

*name* Specifies the name of the assignment to get the value of.

##### Example

```
get_instantiation_assignment_value embeddedsw.configuration.exceptionOffset
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489





#### 9.19.7.4.6 `get_instantiation_assignments`

##### Description

Gets the assignment names in the loaded instantiation.

##### Usage

```
get_instantiation_assignments
```

##### Returns

*String[]* The list of assignment names.

##### Arguments

No arguments

##### Example

```
get_instantiation_assignments
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.7 `get_instantiation_hdl_file_properties`

##### Description

Returns the list of properties in an HDL file associated with an instantiation.

##### Usage

```
get_instantiation_hdl_file_properties
```

##### Returns

*String[]* The list of property names.

##### Arguments

No arguments

##### Example

```
get_instantiation_hdl_file_properties
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.8 `get_instantiation_hdl_file_property`

##### Description

Returns the property value of an HDL file associated with the loaded instantiation.

##### Usage

```
get_instantiation_hdl_file_property <file> <property>
```

##### Returns

*various* The property value.

##### Arguments

*file* Specifies the HDL file name.

*property* Specifies the property name. Refer to *Instantiation Hdl File Properties*.

##### Example

```
get_instantiation_hdl_file_property my_nios2_gen2.vhdl OUTPUT_PATH
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Instantiation Hdl File Properties](#) on page 596



#### 9.19.7.4.9 get\_instantiation\_hdl\_files

##### Description

Returns the list of HDL files of the loaded instantiation.

##### Usage

```
get_instantiation_hdl_files [<kind>]
```

##### Returns

*String[]* The list of HDL file names.

##### Arguments

*kind (optional)* Specifies the file set kind to get the files of. If you do not specify this option, the command gets the QUARTUS\_SYNTH files. Refer to *File Set Kind*.

##### Example

```
get_instantiation_hdl_files quartus_synth
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [File Set Kind](#) on page 594



#### 9.19.7.4.10 `get_instantiation_interface_assignment_value`

##### Description

Gets the assignment value of the loaded instantiation's interface.

##### Usage

```
get_instantiation_interface_assignment_value <interface> <name>
```

##### Returns

*String* The assignment value

##### Arguments

*interface* Specifies the interface name.

*name* Specifies the assignment name to get the value of.

##### Example

```
get_instantiation_interface_assignment_value avs_s0  
embeddedsw.configuration.exceptionOffset
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.11 `get_instantiation_interface_assignments`

##### Description

Gets the assignment names of the loaded instantiation's interface.

##### Usage

```
get_instantiation_interface_assignments <interface>
```

##### Returns

*String[]* The list of assignment names.

##### Arguments

*interface* Specifies the interface name.

##### Example

```
get_instantiation_interface_assignments avs_s0
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.12 `get_instantiation_interface_parameter_value`

##### Description

Returns the parameter value of a loaded instantiation's interface.

##### Usage

```
get_instantiation_interface_parameter_value <interface> <parameter>
```

##### Returns

*String* The parameter value.

##### Arguments

*interface* Specifies the interface name.

*parameter* Specifies the parameter name.

##### Example

```
get_instantiation_interface_parameter_value avs_s0 associatedClock
```

##### Related Links

- [get\\_instantiation\\_interface\\_parameters](#) on page 472
- [set\\_instantiation\\_interface\\_parameter\\_value](#) on page 493
- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.13 `get_instantiation_interface_parameters`

##### Description

Returns the list of parameters of an instantiation's interface.

##### Usage

```
get_instantiation_interface_parameters <interface>
```

##### Returns

*String[]* The list of parameter names.

##### Arguments

*interface* Specifies the interface name.

##### Example

```
get_instantiation_interface_parameters avs_s0
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [get\\_instantiation\\_interface\\_parameter\\_value](#) on page 471
- [set\\_instantiation\\_interface\\_parameter\\_value](#) on page 493





#### 9.19.7.4.14 `get_instantiation_interface_port_properties`

##### Description

Returns the list of port properties of an instantiation's interface.

##### Usage

```
get_instantiation_interface_port_properties
```

##### Returns

*String[]* The list of port properties.

##### Arguments

No arguments

##### Example

```
get_instantiation_interface_port_properties
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.15 `get_instantiation_interface_port_property`

##### Description

Returns the port property value of a loaded instantiation's interface.

##### Usage

```
get_instantiation_interface_port_property <interface> <port>  
<property>
```

##### Returns

*various* The property value.

##### Arguments

*interface* Specifies the interface name.

*port* Specifies the port name.

*property* Specifies the property name. Refer to *Port Properties*.

##### Example

```
get_instantiation_interface_port_property avs_s0 avs_s0_address WIDTH
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Port Properties](#) on page 600



#### 9.19.7.4.16 `get_instantiation_interface_ports`

##### Description

Returns the list of ports of the loaded instantiation's interface.

##### Usage

```
get_instantiation_interface_ports <interface>
```

##### Returns

*String[]* The list of port names.

##### Arguments

*interface* Specifies the interface name.

##### Example

```
get_instantiation_interface_ports avs_s0
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.17 `get_instantiation_interface_property`

##### Description

Returns the value of a single interface property from the specified instantiation interface.

##### Usage

```
get_instantiation_interface_property <interface> <property>
```

##### Returns

*various* The property value.

##### Arguments

*interface* The interface name on the currently loaded interface.

*property* The property name. Refer to *Instantiation Interface Properties*.

##### Example

```
get_instantiation_interface_property in_clk TYPE
```

##### Related Links

- [get\\_instantiation\\_interface\\_properties](#) on page 477
- [load\\_instantiation](#) on page 485
- [Instantiation Interface Properties](#) on page 598



#### 9.19.7.4.18 `get_instantiation_interface_properties`

##### Description

Returns the names of all the available instantiation interface properties, common to all interface types.

##### Usage

```
get_instantiation_interface_properties
```

##### Returns

*String[]* A list of instantiation interface properties.

##### Arguments

No arguments.

##### Example

```
get_instantiation_interface_properties
```

##### Related Links

[get\\_instantiation\\_interface\\_property](#) on page 476



#### 9.19.7.4.19 `get_instantiation_interface_sysinfo_parameter_value`

##### Description

Gets the system info parameter value for a loaded instantiation's interface.

##### Usage

```
get_instantiation_interface_sysinfo_parameter_value <interface>  
<parameter>
```

##### Returns

*various* The system info property value.

##### Arguments

*interface* Specifies the interface name.

*parameter* Specifies the system info parameter name. Refer to *System Info Type*.

##### Example

```
get_instantiation_interface_sysinfo_parameter_value debug_mem_slave  
max_slave_data_width
```

##### Related Links

- [get\\_instantiation\\_interface\\_sysinfo\\_parameters](#) on page 479
- [set\\_instantiation\\_interface\\_sysinfo\\_parameter\\_value](#) on page 495
- [System Info Type Properties](#) on page 589



#### 9.19.7.4.20 `get_instantiation_interface_sysinfo_parameters`

##### Description

Returns the list of system info parameters for the loaded instantiation's interface.

##### Usage

```
get_instantiation_interface_sysinfo_parameters <interface> [<type>]
```

##### Returns

*String[]* The list of system info parameter names.

##### Arguments

*interface* Specifies the interface name.

*type (optional)* Specifies the parameters type to return. If you do not specify this option, the command returns all the parameters. Refer to *Access Type*.

##### Example

```
get_instantiation_interface_sysinfo_parameters debug_mem_slave
```

##### Related Links

- [get\\_instantiation\\_interface\\_sysinfo\\_parameter\\_value](#) on page 478
- [set\\_instantiation\\_interface\\_sysinfo\\_parameter\\_value](#) on page 495
- [Access Type](#) on page 595



#### 9.19.7.4.21 `get_instantiation_interfaces`

##### Description

Returns the list of interfaces for the loaded instantiation.

##### Usage

```
get_instantiation_interfaces
```

##### Returns

*String[]* The list of interface names.

##### Arguments

No arguments.

##### Example

```
get_instantiation_interfaces
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489





#### 9.19.7.4.22 `get_instantiation_properties`

##### Description

Returns the list of properties for the loaded instantiation.

##### Usage

```
get_instantiation_properties
```

##### Returns

*String[]* The list of property names.

##### Arguments

No arguments.

##### Example

```
get_instantiation_properties
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



### 9.19.7.4.23 get\_instantiation\_property

#### Description

Returns the value of the specified property for the loaded instantiation.

#### Usage

```
get_instantiation_property <property>
```

#### Returns

*various* The value of an instantiation property.

#### Arguments

*property* Specifies the property name to get the value of. Refer to *Instantiation Properties*.

#### Example

```
get_instantiation_property HDL_ENTITY_NAME
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Instantiation Properties](#) on page 599



#### 9.19.7.4.24 `get_loaded_instantiation`

##### Description

Returns the instance name of the loaded instantiation.

##### Usage

```
get_loaded_instantiation
```

##### Returns

*String* The instance name.

##### Arguments

No arguments

##### Example

```
get_loaded_instantiation
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.25 import\_instantiation\_interfaces

##### Description

Sets the interfaces of a loaded instantiation, by importing the interfaces from the specified file.

##### Usage

```
import_instantiation_interfaces <file>
```

##### Returns

No return value

##### Arguments

*file* Specifies the The IP or IP-XACT file to import the interfaces from.

##### Example

```
import_instantiation_interfaces ip/my_system/my_system_nios2_gen2_0.ip
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.26 load\_instantiation

##### Description

Loads the instantiation of an instance, so that you can modify the instantiation if necessary.

##### Usage

```
load_instantiation <instance>
```

##### Returns

*boolean* 1 if successful; 0 if unsuccessful.

##### Arguments

*instance* Specifies the instance name.

##### Example

```
load_instantiation cpu
```

##### Related Links

[save\\_instantiation](#) on page 489



#### 9.19.7.4.27 remove\_instantiation\_hdl\_file

##### Description

Removes an HDL file from the loaded instantiation.

##### Usage

```
remove_instantiation_hdl_file <file> [<kind>]
```

##### Returns

No return value.

##### Arguments

*file* Specifies the HDL file name.

*kind (optional)* Specifies the kind of file set to remove the file from. If you do not specify this option, the command removes the file from all the file sets. Refer to *File Set Kind*.

##### Example

```
remove_instantiation_hdl_file my_nios2_gen2.vhdl quartus_synth
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [File Set Kind](#) on page 594



#### 9.19.7.4.28 remove\_instantiation\_interface

##### Description

Removes an interface from a loaded instantiation.

##### Usage

```
remove_instantiation_interface <interface>
```

##### Returns

No return value

##### Arguments

*interface* Specifies the interface name.

##### Example

```
remove_instantiation_interface avs_s0
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



#### 9.19.7.4.29 `remove_instantiation_interface_port`

##### Description

Removes a port from a loaded instantiation's interface.

##### Usage

```
remove_instantiation_interface_port <interface> <port>
```

##### Returns

No return value

##### Arguments

*interface* Specifies the interface name.

*port* Specifies the port name.

##### Example

```
remove_instantiation_interface_port avs_s0 avs_s0_address
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489





#### 9.19.7.4.30 save\_instantiation

##### Description

Saves the loaded instantiation.

##### Usage

```
save_instantiation
```

##### Returns

No return value

##### Arguments

No arguments

##### Example

```
save_instantiation
```

##### Related Links

[load\\_instantiation](#) on page 485



#### 9.19.7.4.31 set\_instantiation\_assignment\_value

##### Description

Sets the assignment value for the loaded instantiation.

##### Usage

```
set_instantiation_assignment_value <name> [<value>]
```

##### Returns

No return value

##### Arguments

*instance* Specifies the assignment name to set value for.

*value (optional)* Specifies the assignment value. If you do not specify this option, the command removes the assignment.

##### Example

```
set_instantiation_assignment_value embeddedsw.configuration.exceptionOffset 32
```

##### Related Links

[get\\_instantiation\\_assignment\\_value](#) on page 464



#### 9.19.7.4.32 set\_instantiation\_hdl\_file\_property

##### Description

Sets the property value for an HDL file associated with a loaded instantiation.

##### Usage

```
set_instantiation_hdl_file_property<file> <property> <value>
```

##### Returns

No return value

##### Arguments

*file* Specifies the HDL file name.

*property* Specifies the property name. Refer to *Instantiation Hdl File Properties*.

*value* Specifies the property value.

##### Example

```
set_instantiation_hdl_file_property my_nios2_gen2.vhdl OUTPUT_PATH  
my_nios2_gen2.vhdl
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Instantiation Hdl File Properties](#) on page 596



### 9.19.7.4.33 set\_instantiation\_interface\_assignment\_value

#### Description

Sets the assignment value for the loaded instantiation's interface.

#### Usage

```
set_instantiation_interface_assignment_value <interface> <name>  
[<value>]
```

#### Returns

No return value

#### Arguments

*interface* Specifies the interface name.

*name* Specifies the assignment name to set the value of.

*value (optional)* Specifies the new assignment value. If you do not specify this value, the command removes the assignment.

#### Example

```
set_instantiation_interface_assignment_value  
embeddedsw.configuration.exceptionOffset 32
```

#### Related Links

[get\\_instantiation\\_assignment\\_value](#) on page 464



#### 9.19.7.4.34 `set_instantiation_interface_parameter_value`

##### Description

Sets the parameter value for the loaded instantiation's interface.

##### Usage

```
set_instantiation_interface_parameter_value <interface> <parameter>  
<value>
```

##### Returns

No return value

##### Arguments

*instance* Specifies the interface name.

*parameter* Specifies the parameter name.

*value* Specifies the parameter value.

##### Example

```
set_instantiation_interface_parameter avs_s0 associatedClock clk
```

##### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [get\\_instantiation\\_interface\\_parameter\\_value](#) on page 471
- [get\\_instantiation\\_interface\\_parameters](#) on page 472



### 9.19.7.4.35 set\_instantiation\_interface\_port\_property

#### Description

Sets the port property value on a loaded instantiation's interface.

#### Usage

```
set_instantiation_interface_port_property <interface> <port>  
<property> <value>
```

#### Returns

No return value

#### Arguments

*interface* Specifies the interface name.

*port* Specifies the port name.

*property* Specifies the property name. Refer to *Port Properties*.

*value* Specifies the property value.

#### Example

```
set_instantiation_interface_port_property avs_s0 avs_s0_address WIDTH 1
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Port Properties](#) on page 600



#### 9.19.7.4.36 set\_instantiation\_interface\_sysinfo\_parameter\_value

##### Description

Sets the system info parameter value for the loaded instantiation's interface.

##### Usage

```
set_instantiation_interface_sysinfo_parameter_value <interface>  
<parameter> <value>
```

##### Returns

No return value

##### Arguments

*interface* Specifies the interface name.

*parameter* Specifies the system info parameter name. Refer to *System Info Type*.

*value* Specifies the system info parameter value.

##### Example

```
set_instantiation_interface_sysinfo_parameter_value debug_mem_slave  
max_slave_data_width 64
```

##### Related Links

- [get\\_instantiation\\_interface\\_sysinfo\\_parameter\\_value](#) on page 478
- [get\\_instantiation\\_interface\\_sysinfo\\_parameters](#) on page 479
- [System Info Type Properties](#) on page 589



### 9.19.7.4.37 set\_instantiation\_property

#### Description

Sets the property value for the loaded instantiation.

#### Usage

```
set_instantiation_property <property> <value>
```

#### Returns

No return value

#### Arguments

*property* Specifies the property name. Refer to *Instantiation Properties*.

*value* Specifies the value to set.

#### Example

```
set_instantiation_property HDL_ENTITY_NAME my_system_nios2_gen2_0
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [Instantiation Properties](#) on page 599





### 9.19.7.5 Components

This section lists the commands that allow you to manipulate the loaded IP components in your Platform Designer system.

- [apply\\_component\\_preset](#) on page 498
- [get\\_component\\_assignment](#) on page 499
- [get\\_component\\_assignments](#) on page 500
- [get\\_component\\_documentation\\_links](#) on page 501
- [get\\_component\\_interface\\_assignment](#) on page 502
- [get\\_component\\_interface\\_assignments](#) on page 503
- [get\\_component\\_interface\\_parameter\\_property](#) on page 504
- [get\\_component\\_interface\\_parameter\\_value](#) on page 505
- [get\\_component\\_interface\\_parameters](#) on page 506
- [get\\_component\\_interface\\_port\\_property](#) on page 507
- [get\\_component\\_interface\\_ports](#) on page 508
- [get\\_component\\_interface\\_property](#) on page 509
- [get\\_component\\_interfaces](#) on page 510
- [get\\_component\\_parameter\\_property](#) on page 511
- [get\\_component\\_parameter\\_value](#) on page 512
- [get\\_component\\_parameters](#) on page 513
- [get\\_component\\_project\\_properties](#) on page 514
- [get\\_component\\_project\\_property](#) on page 515
- [get\\_component\\_property](#) on page 516
- [get\\_loaded\\_component](#) on page 517
- [load\\_component](#) on page 518
- [reload\\_component\\_footprint](#) on page 519
- [save\\_component](#) on page 520
- [set\\_component\\_parameter\\_value](#) on page 521
- [set\\_component\\_project\\_property](#) on page 522



### 9.19.7.5.1 apply\_component\_preset

#### Description

Applies the settings in a preset to the loaded component.

#### Usage

```
apply_component_preset <preset_name>
```

#### Returns

No return value

#### Arguments

*preset\_name* Specifies the preset name.

#### Example

```
apply_component_preset "Custom Debug Settings"
```

#### Related Links

- [load\\_component](#) on page 518
- [set\\_component\\_parameter\\_value](#) on page 521



### 9.19.7.5.2 `get_component_assignment`

#### Description

Returns the assignment value for the loaded component.

#### Usage

```
get_component_assignment <assignment>
```

#### Returns

*String* The specified assignment value.

#### Arguments

*assignment* Specifies the assignment key value to query.

#### Example

```
get_component_assignment embeddedsw.CMacro.colorSpace
```

#### Related Links

- [load\\_component](#) on page 518
- [get\\_component\\_assignments](#) on page 500



### 9.19.7.5.3 `get_component_assignments`

#### Description

Returns the list of assignment keys for the loaded component.

#### Usage

```
get_component_assignments
```

#### Returns

*String[]* The list of assignment keys.

#### Arguments

No arguments

#### Example

```
get_component_assignments
```

#### Related Links

- [get\\_instance\\_assignment](#) on page 432
- [load\\_component](#) on page 518



#### 9.19.7.5.4 `get_component_documentation_links`

##### Description

Returns the list of all documentation links that the loaded component provides.

##### Usage

```
get_component_documentation_links
```

##### Returns

*String[]* The list of documentation links.

##### Arguments

No arguments

##### Example

```
get_component_documentation_links
```

##### Related Links

[load\\_component](#) on page 518



#### 9.19.7.5.5 `get_component_interface_assignment`

##### Description

Returns the assignment value of an interface of the loaded component.

##### Usage

```
get_component_interface_assignment <interface> <assignment>
```

##### Returns

*String* The specified assignment value.

##### Arguments

*interface* Specifies the interface name.

*assignment* Specifies the assignment key to the query.

##### Example

```
get_component_interface_assignment s1 embeddedsw.configuration.isFlash
```

##### Related Links

- [get\\_component\\_interface\\_assignments](#) on page 503
- [load\\_component](#) on page 518



#### 9.19.7.5.6 `get_component_interface_assignments`

##### Description

Returns the list of assignment keys for any assignments that you define for an interface on the loaded component.

##### Usage

```
get_component_interface_assignments <interface>
```

##### Returns

*String[]* The list of assignment keys.

##### Arguments

*interface* Specifies the interface name.

##### Example

```
get_component_interface_assignments s1
```

##### Related Links

- [get\\_component\\_interface\\_assignment](#) on page 502
- [load\\_component](#) on page 518



### 9.19.7.5.7 `get_component_interface_parameter_property`

#### Description

Returns the property value of a parameter in a loaded component's interface. Parameter properties are metadata about how the Intel Quartus Prime uses the parameters.

#### Usage

```
get_component_interface_parameter_property <interface> <parameter>  
<property>
```

#### Returns

*various* The parameter property value.

#### Arguments

*interface* Specifies the interface name.

*parameter* Specifies the parameter name.

*property* Specifies the parameter property. Refer to *Parameter Properties*.

#### Example

```
get_component_interface_parameter_property s0 setupTime ENABLED
```

#### Related Links

- [get\\_component\\_interface\\_parameters](#) on page 506
- [get\\_component\\_interfaces](#) on page 510
- [load\\_component](#) on page 518
- [Parameter Properties](#) on page 583
- [get\\_parameter\\_properties](#) on page 566





### 9.19.7.5.8 `get_component_interface_parameter_value`

#### Description

Returns the parameter value of an interface of the loaded component.

#### Usage

```
get_component_interface_parameter_value <interface> <parameter>
```

#### Returns

*various* The parameter value.

#### Arguments

*interface* Specifies the interface name.

*parameter* Specifies the parameter name.

#### Example

```
get_component_interface_parameter_value s0 setupTime
```

#### Related Links

- [get\\_component\\_interface\\_parameters](#) on page 506
- [get\\_component\\_interfaces](#) on page 510
- [load\\_component](#) on page 518



#### 9.19.7.5.9 `get_component_interface_parameters`

##### Description

Returns the list of parameters for an interface of the loaded component.

##### Usage

```
get_component_interface_parameters <interface>
```

##### Returns

*String[]* The list of parameter names.

##### Arguments

*interface* Specifies the interface name.

##### Example

```
get_component_interface_parameters s0
```

##### Related Links

- [get\\_component\\_interface\\_parameter\\_value](#) on page 505
- [get\\_component\\_interfaces](#) on page 510
- [load\\_component](#) on page 518



#### 9.19.7.5.10 `get_component_interface_port_property`

##### Description

Returns the property value of a port in the interface of the loaded component.

##### Usage

```
get_component_interface_port_property <interface> <port> <property>
```

##### Returns

*various* The port property value

##### Arguments

*interface* Specifies the interface name.

*port* Specifies the port name of the interface.

*property* Specifies the property name of the port. Refer to *Port Properties*.

##### Example

```
get_component_interface_port_property exports tx WIDTH
```

##### Related Links

- [get\\_component\\_interface\\_ports](#) on page 508
- [load\\_component](#) on page 518
- [Port Properties](#) on page 600
- [get\\_port\\_properties](#) on page 544



#### 9.19.7.5.11 `get_component_interface_ports`

##### Description

Returns the list of interface ports of the loaded component.

##### Usage

```
get_component_interface_ports <interface>
```

##### Returns

*String[]* The list of port names

##### Arguments

*interface* Specifies the interface name.

##### Example

```
get_component_interface_ports s0
```

##### Related Links

- [get\\_component\\_interface\\_port\\_property](#) on page 507
- [get\\_component\\_interfaces](#) on page 510
- [load\\_component](#) on page 518



#### 9.19.7.5.12 `get_component_interface_property`

##### Description

Returns the value of a single property from the specified interface for the loaded component.

##### Usage

```
get_component_interface_property <interface> <property>
```

##### Returns

*String* The property value.

##### Arguments

*interface* Specifies the interface name.

*property* Specifies the property name. Refer to *Element Properties*.

##### Example

```
get_interface_property clk_in DISPLAY_NAME
```

##### Related Links

- [load\\_component](#) on page 518
- [Element Properties](#) on page 578
- [get\\_interface\\_properties](#) on page 541



### 9.19.7.5.13 `get_component_interfaces`

#### Description

Returns the list of interfaces in the loaded component.

#### Usage

```
get_component_interfaces
```

#### Returns

*String[]* The list of interface names.

#### Arguments

No arguments

#### Example

```
get_component_interfaces
```

#### Related Links

- [get\\_component\\_interface\\_ports](#) on page 508
- [get\\_component\\_interface\\_property](#) on page 509
- [load\\_component](#) on page 518



#### 9.19.7.5.14 `get_component_parameter_property`

##### Description

Returns the property value of a parameter in the loaded component.

##### Usage

```
get_component_parameter_property <parameter> <property>
```

##### Returns

*Various* The parameter property value.

##### Arguments

*parameter* Specifies the parameter name in the component.

*property* Specifies the property name of the parameter. Refer to *Parameter Properties*.

##### Example

```
get_component_parameter_property baudRate ENABLED
```

##### Related Links

- [get\\_component\\_parameters](#) on page 513
- [get\\_parameter\\_properties](#) on page 566
- [load\\_component](#) on page 518
- [Parameter Properties](#) on page 583



### 9.19.7.5.15 `get_component_parameter_value`

#### Description

Returns the parameter value in the loaded component.

#### Usage

```
get_component_parameter_value <parameter>
```

#### Returns

*various* The parameter value

#### Arguments

*parameter* Specifies the parameter name in the component.

#### Example

```
get_component_parameter_value baudRate
```

#### Related Links

- [get\\_component\\_parameters](#) on page 513
- [load\\_component](#) on page 518
- [set\\_component\\_parameter\\_value](#) on page 521





#### 9.19.7.5.16 `get_component_parameters`

##### Description

Returns the list of parameters in the loaded component.

##### Usage

```
get_component_parameters
```

##### Returns

*String[]* The list of parameters in the component.

##### Arguments

No arguments

##### Example

```
get_instance_parameters
```

##### Related Links

- [get\\_component\\_parameter\\_property](#) on page 511
- [get\\_component\\_parameter\\_value](#) on page 512
- [load\\_component](#) on page 518
- [set\\_component\\_parameter\\_value](#) on page 521



### 9.19.7.5.17 `get_component_project_properties`

#### Description

Returns the list of properties that you query about the loaded component's Intel Quartus Prime project.

#### Usage

```
get_component_project_properties
```

#### Returns

*String[]* The list of project properties.

#### Arguments

No arguments

#### Example

```
get_component_project_properties
```

#### Related Links

- [get\\_component\\_project\\_property](#) on page 515
- [load\\_component](#) on page 518
- [set\\_component\\_project\\_property](#) on page 522



### 9.19.7.5.18 `get_component_project_property`

#### Description

Returns the project property value of the loaded component. Only select project properties are available.

#### Usage

```
get_component_project_property <property>
```

#### Returns

*String* The property value.

#### Arguments

*property* Specifies the project property name. Refer to *Project Properties*.

#### Example

```
get_component_project_property HIDE_FROM_IP_CATALOG
```

#### Related Links

- [get\\_component\\_project\\_properties](#) on page 514
- [load\\_component](#) on page 518
- [set\\_component\\_project\\_property](#) on page 522
- [Project Properties](#) on page 588



### 9.19.7.5.19 `get_component_property`

#### Description

Returns the property value of the loaded component.

#### Usage

```
get_component_property <property>
```

#### Returns

*String* The property value.

#### Arguments

*property* The property name on the loaded component. Refer to *Element Properties*.

#### Example

```
get_component_property CLASS_NAME
```

#### Related Links

- [load\\_component](#) on page 518
- [get\\_instance\\_properties](#) on page 450
- [Element Properties](#) on page 578



#### 9.19.7.5.20 `get_loaded_component`

##### Description

Returns the instance name associated with the loaded component.

##### Usage

```
get_loaded_component
```

##### Returns

*String* The instance name.

##### Arguments

No arguments

##### Example

```
get_loaded_component
```

##### Related Links

- [load\\_component](#) on page 518
- [save\\_component](#) on page 520



### 9.19.7.5.21 load\_component

#### Description

Loads the actual component inside of a generic component, so that you can modify the component parameters.

#### Usage

```
load_component <instance>
```

#### Returns

*boolean* 1 if successful; 0 if unsuccessful.

#### Arguments

*instance* Specifies the instance name.

#### Example

```
load_component cpu
```

#### Related Links

- [get\\_loaded\\_component](#) on page 517
- [save\\_component](#) on page 520



### 9.19.7.5.22 reload\_component\_footprint

#### Description

Validates the footprint of a specified child instance, and updates the footprint of the instance in case of issues.

#### Usage

```
reload_component_footprint [<instance>]
```

#### Returns

*String[]* A list of validation messages.

#### Arguments

<i>instance</i> ( <i>optional</i> )	Specifies the child instance name to validate. If you do not specify this option, the command validates all the generic components in the system.
----------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------

#### Example

```
reload_component_footprint cpu_0
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [validate\\_component\\_footprint](#) on page 558



### 9.19.7.5.23 save\_component

#### Description

Saves the loaded component.

#### Usage

save\_component

#### Returns

No return value

#### Arguments

No arguments

#### Example

```
save_component
```

#### Related Links

- [get\\_loaded\\_component](#) on page 517
- [load\\_component](#) on page 518





#### 9.19.7.5.24 set\_component\_parameter\_value

##### Description

Sets the parameter value for the loaded component.

##### Usage

```
set_component_parameter_value <parameter> <value>
```

##### Returns

No return value

##### Arguments

*parameter* Specifies the parameter name.

*parameter* Specifies the new parameter value.

##### Example

```
set_component_parameter_value baudRate 9600
```

##### Related Links

- [get\\_component\\_parameter\\_value](#) on page 512
- [get\\_component\\_parameters](#) on page 513
- [load\\_component](#) on page 518



### 9.19.7.5.25 set\_component\_project\_property

#### Description

Sets the project property value of the loaded component, such as hiding from the IP catalog.

#### Usage

```
set_component_project_property <property> <value>
```

#### Returns

No return value

#### Arguments

*property* Specifies the property name. Refer to *Project Properties*.

*value* Specifies the new property value.

#### Example

```
set_component_project_property HIDE_FROM_IP_CATALOG false
```

#### Related Links

- [get\\_component\\_project\\_properties](#) on page 514
- [get\\_component\\_project\\_property](#) on page 515
- [load\\_component](#) on page 518
- [Project Properties](#) on page 588



### 9.19.7.6 Connections

This section lists the commands that allow you to manipulate the interface connections in your Platform Designer system.

[add\\_connection](#) on page 524

[auto\\_connect](#) on page 525

[get\\_connection\\_parameter\\_property](#) on page 526

[get\\_connection\\_parameter\\_value](#) on page 527

[get\\_connection\\_parameters](#) on page 528

[get\\_connection\\_properties](#) on page 529

[get\\_connection\\_property](#) on page 530

[get\\_connections](#) on page 531

[remove\\_connection](#) on page 532

[remove\\_dangling\\_connections](#) on page 533

[set\\_connection\\_parameter\\_value](#) on page 534

### 9.19.7.6.1 add\_connection

#### Description

Connects the named interfaces using an appropriate connection type. Both interface names consist of an instance name, followed by the interface name that the module provides.

#### Usage

```
add_connection <start> [<end>]
```

#### Returns

No return value.

#### Arguments

*start* The start interface that you connect, in `<instance_name>.<interface_name>` format. If you do not specify the end argument, the connection must be of the form `<instance1>.<interface>/<instance2>.<interface>`.

*end (optional)* The end interface that you connect, in `<instance_name>.<interface_name>` format.

#### Example

```
add_connection dma.read_master sdram.sl
```

#### Related Links

- [get\\_connection\\_parameter\\_value](#) on page 527
- [get\\_connection\\_property](#) on page 530
- [get\\_connections](#) on page 531
- [remove\\_connection](#) on page 532
- [set\\_connection\\_parameter\\_value](#) on page 534



### 9.19.7.6.2 auto\_connect

#### Description

Creates connections from an instance or instance interface to matching interfaces of other instances in the system. For example, Avalon-MM slaves connect to Avalon-MM masters.

#### Usage

`auto_connect <element>`

#### Returns

No return value.

#### Arguments

*element* The instance interface name, or the instance name.

#### Example

```
auto_connect sdram  
auto_connect uart_0.s1
```

#### Related Links

[add\\_connection](#) on page 524



### 9.19.7.6.3 `get_connection_parameter_property`

#### Description

Returns the property value of a parameter in a connection. Parameter properties are metadata about how Platform Designer uses the parameter.

#### Usage

```
get_connection_parameter_property <connection> <parameter> <property>
```

#### Returns

*various* The parameter property value.

#### Arguments

*connection* The connection to query.

*parameter* The parameter name.

*property* The property of the connection. Refer to *Parameter Properties*.

#### Example

```
get_connection_parameter_property cpu.data_master/dma0.csr baseAddress UNITS
```

#### Related Links

- [get\\_connection\\_parameter\\_value](#) on page 527
- [get\\_connection\\_property](#) on page 530
- [get\\_connections](#) on page 531
- [get\\_parameter\\_properties](#) on page 566
- [Parameter Properties](#) on page 583



#### 9.19.7.6.4 `get_connection_parameter_value`

##### Description

Returns the parameter value of the connection. Parameters represent aspects of the connection that you can modify, such as the base address for an Avalon-MM connection.

##### Usage

```
get_connection_parameter_value <connection> <parameter>
```

##### Returns

*various* The parameter value.

##### Arguments

*connection* The connection to query.

*parameter* The parameter name.

##### Example

```
get_connection_parameter_value cpu.data_master/dma0.csr baseAddress
```

##### Related Links

- [get\\_connection\\_parameters](#) on page 528
- [get\\_connections](#) on page 531
- [set\\_connection\\_parameter\\_value](#) on page 534



### 9.19.7.6.5 get\_connection\_parameters

#### Description

Returns the list of parameters of a connection.

#### Usage

```
get_connection_parameters <connection>
```

#### Returns

*String[]* The list of parameter names.

#### Arguments

*connection* The connection to query.

#### Example

```
get_connection_parameters cpu.data_master/dma0.csr
```

#### Related Links

- [get\\_connection\\_parameter\\_property](#) on page 526
- [get\\_connection\\_parameter\\_value](#) on page 527
- [get\\_connection\\_property](#) on page 530





#### 9.19.7.6.6 `get_connection_properties`

##### Description

Returns the properties list of a connection.

##### Usage

```
get_connection_properties
```

##### Returns

*String[]* The list of connection properties.

##### Arguments

No arguments.

##### Example

```
get_connection_properties
```

##### Related Links

- [get\\_connection\\_property](#) on page 530
- [get\\_connections](#) on page 531



### 9.19.7.6.7 `get_connection_property`

#### Description

Returns the property value of a connection. Properties represent aspects of the connection that you can modify, such as the connection type.

#### Usage

```
get_connection_property <connection> <property>
```

#### Returns

*String* The connection property value.

#### Arguments

*connection* The connection to query.

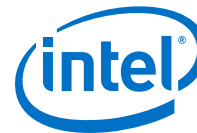
*property* The connection property name. Refer to *Connection Properties*.

#### Example

```
get_connection_property cpu.data_master/dma0.csr TYPE
```

#### Related Links

- [get\\_connection\\_properties](#) on page 529
- [Connection Properties](#) on page 575



### 9.19.7.6.8 get\_connections

#### Description

Returns the list of all connections in the system if you do not specify any element. If you specify a child instance, for example `cpu`, Platform Designer returns all connections to any interface on the instance. If you specify an interface of a child instance, for example `cpu.instruction_master`, Platform Designer returns all connections to that interface.

#### Usage

```
get_connections [<element>]
```

#### Returns

*String[]* The list of connections.

#### Arguments

*element (optional)* The child instance name, or the qualified interface name on a child instance.

#### Example

```
get_connections  
get_connections cpu  
get_connections cpu.instruction_master
```

#### Related Links

- [add\\_connection](#) on page 524
- [remove\\_connection](#) on page 532



### 9.19.7.6.9 remove\_connection

#### Description

Removes a connection from the system.

#### Usage

```
remove_connection <connection>
```

#### Returns

No return value.

#### Arguments

*connection* The connection name to remove.

#### Example

```
remove_connection cpu.data_master/sdram.s0
```

#### Related Links

- [add\\_connection](#) on page 524
- [get\\_connections](#) on page 531



#### 9.19.7.6.10 remove\_dangling\_connections

##### Description

Removes connections where both end points of the connection no longer exist in the system.

##### Usage

```
remove_dangling_connections
```

##### Returns

No return value.

##### Arguments

No arguments.

##### Example

```
remove_dangling_connections
```

##### Related Links

- [add\\_connection](#) on page 524
- [get\\_connections](#) on page 531
- [remove\\_connection](#) on page 532



### 9.19.7.6.11 set\_connection\_parameter\_value

#### Description

Sets the parameter value for a connection.

#### Usage

```
set_connection_parameter_value <connection> <parameter> <value>
```

#### Returns

No return value.

#### Arguments

*connection* The connection name.

*parameter* The parameter name.

*value* The new parameter value.

#### Example

```
set_connection_parameter_value cpu.data_master/dma0.csr baseAddress  
"0x000a0000"
```

#### Related Links

- [get\\_connection\\_parameter\\_value](#) on page 527
- [get\\_connection\\_parameters](#) on page 528



### 9.19.7.7 Top-level Exports

This section lists the commands that allow you to manipulate the exported interfaces in your Platform Designer system.

[add\\_interface](#) on page 536

[get\\_exported\\_interface\\_sysinfo\\_parameter\\_value](#) on page 537

[get\\_exported\\_interface\\_sysinfo\\_parameters](#) on page 538

[get\\_interface\\_port\\_property](#) on page 539

[get\\_interface\\_ports](#) on page 540

[get\\_interface\\_properties](#) on page 541

[get\\_interface\\_property](#) on page 542

[get\\_interfaces](#) on page 543

[get\\_port\\_properties](#) on page 544

[remove\\_interface](#) on page 545

[set\\_exported\\_interface\\_sysinfo\\_parameter\\_value](#) on page 546

[set\\_interface\\_port\\_property](#) on page 547

[set\\_interface\\_property](#) on page 548



### 9.19.7.7.1 add\_interface

#### Description

Adds an interface to your system, which Platform Designer uses to export an interface from within the system. You specify the exported internal interface with `set_interface_property <interface> EXPORT_OF instance.interface`.

#### Usage

`add_interface <name> <type> <direction>`.

#### Returns

No return value.

#### Arguments

*name* The name of the interface that Platform Designer exports from the system.

*type* The type of interface.

*direction* The interface direction.

#### Example

```
add_interface my_export conduit end
set_interface_property my_export EXPORT_OF uart_0.external_connection
```

#### Related Links

- [get\\_interface\\_ports](#) on page 540
- [get\\_interface\\_properties](#) on page 541
- [get\\_interface\\_property](#) on page 542
- [set\\_interface\\_property](#) on page 548





### 9.19.7.7.2 get\_exported\_interface\_sysinfo\_parameter\_value

#### Description

Gets the value of a system info parameter for an exported interface.

#### Usage

```
get_exported_interface_sysinfo_parameter_value <interface>  
<parameter>
```

#### Returns

*various* The system info parameter value.

#### Arguments

*interface* Specifies the name of the exported interface.

*parameter* Specifies the name of the system info parameter. Refer to *System Info Type*.

#### Example

```
get_exported_interface_sysinfo_parameter_value clk clock_rate
```

#### Related Links

- [get\\_exported\\_interface\\_sysinfo\\_parameters](#) on page 538
- [set\\_exported\\_interface\\_sysinfo\\_parameter\\_value](#) on page 546
- [System Info Type Properties](#) on page 589



### 9.19.7.7.3 get\_exported\_interface\_sysinfo\_parameters

#### Description

Returns the list of system info parameters for an exported interface.

#### Usage

```
get_exported_interface_sysinfo_parameters <interface> [<type>]
```

#### Returns

*String[]* The list of system info parameter names.

#### Arguments

*interface* Specifies the name of the exported interface.

*type (optional)* Specifies the parameters type to return. If you do not specify this option, the command returns all the parameters. Refer to *Access Type*.

#### Example

```
get_exported_interface_sysinfo_parameters clk
```

#### Related Links

- [get\\_exported\\_interface\\_sysinfo\\_parameter\\_value](#) on page 537
- [set\\_exported\\_interface\\_sysinfo\\_parameter\\_value](#) on page 546
- [Access Type](#) on page 595



#### 9.19.7.7.4 `get_interface_port_property`

##### Description

Returns the value of a property of a port contained by one of the top-level exported interfaces.

##### Usage

```
get_interface_port_property <interface> <port> <property>
```

##### Returns

*various* The property value.

##### Arguments

*interface* The name of a top-level interface of the system.

*port* The port name in the interface.

*property* The property name on the port. Refer to *Port Properties*.

##### Example

```
get_interface_port_property uart_exports tx DIRECTION
```

##### Related Links

- [get\\_interface\\_ports](#) on page 540
- [get\\_port\\_properties](#) on page 544
- [Port Properties](#) on page 587



### 9.19.7.7.5 get\_interface\_ports

#### Description

Returns the names of all the added ports to a given interface.

#### Usage

```
get_interface_ports <interface>
```

#### Returns

*String[]* The list of port names.

#### Arguments

*interface* The top-level interface name of the system.

#### Example

```
get_interface_ports export_clk_out
```

#### Related Links

- [get\\_interface\\_port\\_property](#) on page 539
- [get\\_interfaces](#) on page 543



#### 9.19.7.7.6 `get_interface_properties`

##### Description

Returns the names of all the available interface properties common to all interface types.

##### Usage

```
get_interface_properties
```

##### Returns

*String[]* The list of interface properties.

##### Arguments

No arguments.

##### Example

```
get_interface_properties
```

##### Related Links

- [get\\_interface\\_property](#) on page 542
- [set\\_interface\\_property](#) on page 548



### 9.19.7.7.7 `get_interface_property`

#### Description

Returns the value of a single interface property from the specified interface.

#### Usage

```
get_interface_property <interface> <property>
```

#### Returns

*various* The property value.

#### Arguments

*interface* The name of a top-level interface of the system.

*property* The name of the property. Refer to *Interface Properties*.

#### Example

```
get_interface_property export_clk_out EXPORT_OF
```

#### Related Links

- [get\\_interface\\_properties](#) on page 541
- [set\\_interface\\_property](#) on page 548
- [Interface Properties](#) on page 580



### 9.19.7.7.8 get\_interfaces

#### Description

Returns the list of top-level interfaces in the system.

#### Usage

```
get_interfaces
```

#### Returns

*String[]* The list of the top-level interfaces exported from the system.

#### Arguments

No arguments.

#### Example

```
get_interfaces
```

#### Related Links

- [add\\_interface](#) on page 536
- [get\\_interface\\_ports](#) on page 540
- [get\\_interface\\_property](#) on page 542
- [remove\\_interface](#) on page 545
- [set\\_interface\\_property](#) on page 548



### 9.19.7.7.9 get\_port\_properties

#### Description

Returns the list of properties that you can query for ports.

#### Usage

```
get_port_properties
```

#### Returns

*String[]* The list of port properties.

#### Arguments

No arguments.

#### Example

```
get_port_properties
```

#### Related Links

- [get\\_instance\\_interface\\_port\\_property](#) on page 440
- [get\\_instance\\_interface\\_ports](#) on page 441
- [get\\_instance\\_port\\_property](#) on page 449
- [get\\_interface\\_port\\_property](#) on page 539
- [get\\_interface\\_ports](#) on page 540





#### 9.19.7.7.10 remove\_interface

##### Description

Removes an exported top-level interface from the system.

##### Usage

```
remove_interface <interface>
```

##### Returns

No return value.

##### Arguments

*interface* The name of the exported top-level interface.

##### Example

```
remove_interface clk_out
```

##### Related Links

- [add\\_interface](#) on page 536
- [get\\_interfaces](#) on page 543



### 9.19.7.7.11 set\_exported\_interface\_sysinfo\_parameter\_value

#### Description

Sets the system info parameter value for an exported interface.

#### Usage

```
set_exported_interface_sysinfo_parameter_value <interface>  
<parameter> <value>
```

#### Returns

No return value

#### Arguments

*interface* Specifies the name of the exported interface.

*parameter* Specifies the name of the system info parameter. Refer to *System Info Type*.

*value* Specifies the system info parameter value.

#### Example

```
set_exported_interface_sysinfo_parameter_value clk clock_rate 5000000
```

#### Related Links

- [get\\_exported\\_interface\\_sysinfo\\_parameter\\_value](#) on page 537
- [get\\_exported\\_interface\\_sysinfo\\_parameters](#) on page 538
- [System Info Type Properties](#) on page 589



#### 9.19.7.7.12 set\_interface\_port\_property

##### Description

Sets the port property in a top-level interface of the system.

##### Usage

```
set_interface_port_property <interface> <port> <property> <value>
```

##### Returns

No return value

##### Arguments

*interface* Specifies the top-level interface name of the system.

*port* Specifies the port name in a top-level interface of the system.

*property* Specifies the property name of the port. Refer to *Port Properties*.

*value* Specifies the property value.

##### Example

```
set_interface_port_property clk clk_clk NAME my_clk
```

##### Related Links

- [Port Properties](#) on page 600
- [get\\_interface\\_ports](#) on page 540
- [get\\_interfaces](#) on page 543
- [get\\_port\\_properties](#) on page 544



### 9.19.7.7.13 set\_interface\_property

#### Description

Sets the value of a property on an exported top-level interface. You use this command to set the `EXPORT_OF` property to specify which interface of a child instance is exported via this top-level interface.

#### Usage

```
set_interface_property <interface> <property> <value>
```

#### Returns

No return value.

#### Arguments

*interface* The name of an exported top-level interface.

*property* The name of the property. Refer to *Interface Properties*.

*value* The property value.

#### Example

```
set_interface_property clk_out EXPORT_OF clk.clk_out
```

#### Related Links

- [add\\_interface](#) on page 536
- [get\\_interface\\_properties](#) on page 541
- [get\\_interface\\_property](#) on page 542
- [Interface Properties](#) on page 580



### 9.19.7.8 Validation

This section lists the commands that allow you to validate the components, instances, interfaces and connections in your Platform Designer system.

[set\\_validation\\_property](#) on page 550

[sync\\_sysinfo\\_parameters](#) on page 551

[validate\\_component](#) on page 552

[validate\\_component\\_interface](#) on page 553

[validate\\_connection](#) on page 554

[validate\\_instance](#) on page 555

[validate\\_instance\\_interface](#) on page 556

[validate\\_system](#) on page 557

[validate\\_component\\_footprint](#) on page 558

[reload\\_component\\_footprint](#) on page 519



### 9.19.7.8.1 set\_validation\_property

#### Description

Sets a property that affects how and when validation is run. To disable system validation after each scripting command, set `AUTOMATIC_VALIDATION` to `False`.

#### Usage

```
set_validation_property <property> <value>
```

#### Returns

No return value.

#### Arguments

*property* The name of the property. Refer to *Validation Properties*.

*value* The new property value.

#### Example

```
set_validation_property AUTOMATIC_VALIDATION false
```

#### Related Links

- [validate\\_system](#) on page 557
- [Validation Properties](#) on page 592



### 9.19.7.8.2 sync\_sysinfo\_parameters

#### Description

Updates the system info parameters of the specified generic component.

#### Usage

```
sync_sysinfo_parameters [<instance> ]
```

#### Returns

*String[]* A list of update messages.

#### Arguments

<i>instance</i> ( <i>optional</i> )	Specifies the name of the instance to sync. If you do not specify this option, the command synchronizes all the generic components in the system.
----------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------

#### Example

```
sync_sysinfo_parameters cpu_0
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



### 9.19.7.8.3 validate\_component

#### Description

Validates the loaded component.

#### Usage

validate\_component

#### Returns

*String[]* A list of validation messages.

#### Arguments

No arguments

#### Example

```
validate_component
```

#### Related Links

- [validate\\_component\\_interface](#) on page 553
- [load\\_component](#) on page 518





#### 9.19.7.8.4 validate\_component\_interface

##### Description

Validates an interface of the loaded component.

##### Usage

```
validate_component_interface <interface>
```

##### Returns

*String[]* List of validation messages

##### Arguments

*instance* Specifies the name of the instance for the loaded component.

##### Example

```
validate_instance_interface data_master
```

##### Related Links

- [load\\_component](#) on page 518
- [validate\\_component](#) on page 552



### 9.19.7.8.5 validate\_connection

#### Description

Validates the specified connection and returns validation messages.

#### Usage

```
validate_connection <connection>
```

#### Returns

A list of validation messages.

#### Arguments

*connection* The connection name to validate.

#### Example

```
validate_connection cpu.data_master/sdram.sl
```

#### Related Links

- [validate\\_instance](#) on page 555
- [validate\\_instance\\_interface](#) on page 556
- [validate\\_system](#) on page 557



### 9.19.7.8.6 validate\_instance

#### Description

Validates the specified child instance and returns validation messages.

#### Usage

```
validate_instance <instance>
```

#### Returns

A list of validation messages.

#### Arguments

*instance* The child instance name to validate.

#### Example

```
validate_instance cpu
```

#### Related Links

- [validate\\_connection](#) on page 554
- [validate\\_instance\\_interface](#) on page 556
- [validate\\_system](#) on page 557



### 9.19.7.8.7 `validate_instance_interface`

#### Description

Validates an interface of an instance and returns validation messages.

#### Usage

```
validate_instance_interface <instance> <interface>
```

#### Returns

A list of validation messages.

#### Arguments

*instance* The child instance name.

*interface* The interface to validate.

#### Example

```
validate_instance_interface cpu data_master
```

#### Related Links

- [validate\\_connection](#) on page 554
- [validate\\_instance](#) on page 555
- [validate\\_system](#) on page 557



### 9.19.7.8.8 validate\_system

#### Description

Validates the system and returns validation messages.

#### Usage

```
validate_system
```

#### Returns

A list of validation messages.

#### Arguments

No arguments.

#### Example

```
validate_system
```

#### Related Links

- [validate\\_connection](#) on page 554
- [validate\\_instance](#) on page 555
- [validate\\_instance\\_interface](#) on page 556



### 9.19.7.8.9 validate\_component\_footprint

#### Description

Validates the footprint of the specified child instance.

#### Usage

```
validate_component_footprint <instance>
```

#### Returns

*String[]* List of validation messages.

#### Arguments

*instance (optional)* Specifies the child instance name. If you omit this option, the command validates all generic components in the system.

#### Example

```
validate_component_footprint cpu_0
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489



### 9.19.7.8.10 reload\_component\_footprint

#### Description

Validates the footprint of a specified child instance, and updates the footprint of the instance in case of issues.

#### Usage

```
reload_component_footprint [<instance>]
```

#### Returns

*String[]* A list of validation messages.

#### Arguments

<i>instance</i> ( <i>optional</i> )	Specifies the child instance name to validate. If you do not specify this option, the command validates all the generic components in the system.
----------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------

#### Example

```
reload_component_footprint cpu_0
```

#### Related Links

- [load\\_instantiation](#) on page 485
- [save\\_instantiation](#) on page 489
- [validate\\_component\\_footprint](#) on page 558



### 9.19.7.9 Miscellaneous

This section lists the miscellaneous commands that you can use for your Platform Designer systems.

[auto\\_assign\\_base\\_addresses](#) on page 561

[auto\\_assign\\_irqs](#) on page 562

[auto\\_assign\\_system\\_base\\_addresses](#) on page 563

[get\\_interconnect\\_requirement](#) on page 564

[get\\_interconnect\\_requirements](#) on page 565

[get\\_parameter\\_properties](#) on page 566

[lock\\_avalon\\_base\\_address](#) on page 567

[send\\_message](#) on page 568

[set\\_interconnect\\_requirement](#) on page 569

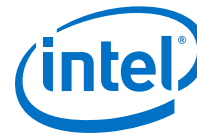
[set\\_use\\_testbench\\_naming\\_pattern](#) on page 570

[unlock\\_avalon\\_base\\_address](#) on page 571

[get\\_testbench\\_dutname](#) on page 572

[get\\_use\\_testbench\\_naming\\_pattern](#) on page 573





### 9.19.7.9.1 auto\_assign\_base\_addresses

#### Description

Assigns base addresses to all memory-mapped interfaces of an instance in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

#### Usage

```
auto_assign_base_addresses <instance>
```

#### Returns

No return value.

#### Arguments

*instance* The name of the instance with memory-mapped interfaces.

#### Example

```
auto_assign_base_addresses sdram
```

#### Related Links

- [auto\\_assign\\_system\\_base\\_addresses](#) on page 563
- [lock\\_avalon\\_base\\_address](#) on page 567
- [unlock\\_avalon\\_base\\_address](#) on page 571



### 9.19.7.9.2 auto\_assign\_irqs

#### Description

Assigns interrupt numbers to all connected interrupt senders of an instance in the system.

#### Usage

```
auto_assign_irqs <instance>
```

#### Returns

No return value.

#### Arguments

*instance* The name of the instance with an interrupt sender.

#### Example

```
auto_assign_irqs uart_0
```



### 9.19.7.9.3 auto\_assign\_system\_base\_addresses

#### Description

Assigns legal base addresses to all memory-mapped interfaces of all instances in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

#### Usage

`auto_assign_system_base_addresses`

#### Returns

No return value.

#### Arguments

No arguments.

#### Example

```
auto_assign_system_base_addresses
```

#### Related Links

- [auto\\_assign\\_base\\_addresses](#) on page 561
- [lock\\_avalon\\_base\\_address](#) on page 567
- [unlock\\_avalon\\_base\\_address](#) on page 571



#### 9.19.7.9.4 get\_interconnect\_requirement

##### Description

Returns the value of an interconnect requirement for a system or interface of a child instance.

##### Usage

```
get_interconnect_requirement <element_id> <requirement>
```

##### Returns

*String* The value of the interconnect requirement.

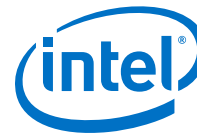
##### Arguments

*element\_id* `{$system}` for the system, or the qualified name of the interface of an instance, in `<instance>.<interface>` format. In Tcl, the system identifier is escaped, for example, `{$system}`.

*requirement* The name of the requirement.

##### Example

```
get_interconnect_requirement {$system} qsys_mm.maxAdditionalLatency
```



### 9.19.7.9.5 get\_interconnect\_requirements

#### Description

Returns the list of all interconnect requirements in the system.

#### Usage

```
get_interconnect_requirements
```

#### Returns

*String[]* A flattened list of interconnect requirements. Every sequence of three elements in the list corresponds to one interconnect requirement. The first element in the sequence is the element identifier. The second element is the requirement name. The third element is the value. You can loop over the returned list with a `foreach` loop, for example:

```
foreach { element_id name value } $requirement_list { loop_body
}
```

#### Arguments

No arguments.

#### Example

```
get_interconnect_requirements
```



### 9.19.7.9.6 `get_parameter_properties`

#### Description

Returns the list of properties that you can query for any parameters, for example parameters of instances, interfaces, instance interfaces, and connections.

#### Usage

```
get_parameter_properties
```

#### Returns

*String[]* The list of parameter properties.

#### Arguments

No arguments.

#### Example

```
get_parameter_properties
```

#### Related Links

- [get\\_connection\\_parameter\\_property](#) on page 526
- [get\\_instance\\_interface\\_parameter\\_property](#) on page 437
- [get\\_instance\\_parameter\\_property](#) on page 445



### 9.19.7.9.7 lock\_avalon\_base\_address

#### Description

Prevents the memory-mapped base address from being changed for connections to the specified interface of an instance when Platform Designer runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

#### Usage

```
lock_avalon_base_address <instance.interface>
```

#### Returns

No return value.

#### Arguments

*instance.interface* The qualified name of the interface of an instance, in `<instance>.<interface>` format.

#### Example

```
lock_avalon_base_address sdram.s1
```

#### Related Links

- [auto\\_assign\\_base\\_addresses](#) on page 561
- [auto\\_assign\\_system\\_base\\_addresses](#) on page 563
- [unlock\\_avalon\\_base\\_address](#) on page 571



### 9.19.7.9.8 send\_message

#### Description

Sends a message to the user of the component. The message text is normally HTML. You can use the `<b>` element to provide emphasis. If you do not want the message text to be HTML, then pass a list like `{ Info Text }` as the message level,

#### Usage

```
send_message <level> <message>
```

#### Returns

No return value.

#### Arguments

*level* Intel Quartus Prime supports the following message levels:

- ERROR—provides an error message.
- WARNING—provides a warning message.
- INFO—provides an informational message.
- PROGRESS—provides a progress message.
- DEBUG—provides a debug message when debug mode is enabled.

*message* The text of the message.

#### Example

```
send_message ERROR "The system is down!"  
send_message { Info Text } "The system is up!"
```





### 9.19.7.9.9 set\_interconnect\_requirement

#### Description

Sets the value of an interconnect requirement for a system or an interface of a child instance.

#### Usage

```
set_interconnect_requirement <element_id> <requirement> <value>
```

#### Returns

No return value.

#### Arguments

*element\_id* `{$system}` for the system, or qualified name of the interface of an instance, in `<instance>.<interface>` format. In Tcl, the system identifier is escaped, for example, `{$system}`.

*requirement* The name of the requirement.

*value* The requirement value.

#### Example

```
set_interconnect_requirement {$system} qsys_mm.clockCrossingAdapter HANDSHAKE
```



### 9.19.7.9.10 set\_use\_testbench\_naming\_pattern

#### Description

Use this command to create testbench systems so that the generated file names for the test system match the system's original generated file names. Without setting this command, the generated file names for the test system receive the top-level testbench system name.

#### Usage

```
set_use_testbench_naming_pattern <value>
```

#### Returns

No return value.

#### Arguments

*value* True or false.

#### Example

```
set_use_testbench_naming_pattern true
```

#### Notes

Use this command only to create testbench systems.



### 9.19.7.9.11 unlock\_avalon\_base\_address

#### Description

Allows the memory-mapped base address to change for connections to the specified interface of an instance when Platform Designer runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

#### Usage

```
unlock_avalon_base_address <instance.interface>
```

#### Returns

No return value.

#### Arguments

*instance.interface* The qualified name of the interface of an instance, in `<instance>.<interface>` format.

#### Example

```
unlock_avalon_base_address sdram.s1
```

#### Related Links

- [auto\\_assign\\_base\\_addresses](#) on page 561
- [auto\\_assign\\_system\\_base\\_addresses](#) on page 563
- [lock\\_avalon\\_base\\_address](#) on page 567



### 9.19.7.9.12 `get_testbench_dutname`

#### Description

Returns the currently set dutname for the test-bench systems. Use this command only when creating test-bench systems.

#### Usage

```
get_testbench_dutname
```

#### Returns

*String* The currently set dutname. Returns NULL if empty.

#### Arguments

No arguments.

#### Example

```
get_testbench_dutname
```

#### Related Links

- [get\\_use\\_testbench\\_naming\\_pattern](#) on page 573
- [set\\_use\\_testbench\\_naming\\_pattern](#) on page 570



### 9.19.7.9.13 `get_use_testbench_naming_pattern`

#### Description

Verifies if the test-bench naming pattern is set to be used. Use this command only when creating test-bench systems.

#### Usage

```
get_use_testbench_naming_pattern
```

#### Returns

*boolean* True, if the test-bench naming pattern is set to be used.

#### Arguments

No arguments.

#### Example

```
get_use_testbench_naming_pattern
```

#### Related Links

- [get\\_testbench\\_dutname](#) on page 572
- [set\\_use\\_testbench\\_naming\\_pattern](#) on page 570



### 9.19.8 Platform Designer Scripting Property Reference

Interface properties work differently for `_hw.tcl` scripting than with Platform Designer scripting. In `_hw.tcl`, interfaces do not distinguish between properties and parameters. In Platform Designer scripting, the properties and parameters are unique.

The following are the Platform Designer scripting properties:

- [Connection Properties](#) on page 575
- [Design Environment Type Properties](#) on page 576
- [Direction Properties](#) on page 577
- [Element Properties](#) on page 578
- [Instance Properties](#) on page 579
- [Interface Properties](#) on page 580
- [Message Levels Properties](#) on page 581
- [Module Properties](#) on page 582
- [Parameter Properties](#) on page 583
- [Parameter Status Properties](#) on page 585
- [Parameter Type Properties](#) on page 586
- [Port Properties](#) on page 587
- [Project Properties](#) on page 588
- [System Info Type Properties](#) on page 589
- [Units Properties](#) on page 591
- [Validation Properties](#) on page 592
- [Interface Direction](#) on page 593
- [File Set Kind](#) on page 594
- [Access Type](#) on page 595
- [Instantiation Hdl File Properties](#) on page 596
- [Instantiation Interface Duplicate Type](#) on page 597
- [Instantiation Interface Properties](#) on page 598
- [Instantiation Properties](#) on page 599
- [Port Properties](#) on page 600
- [VHDL Type](#) on page 601



### 9.19.8.1 Connection Properties

Type	Name	Description
string	END	Indicates the end interface of the connection.
string	NAME	Indicates the name of the connection.
string	START	Indicates the start interface of the connection.
String	TYPE	The type of the connection.



### 9.19.8.2 Design Environment Type Properties

#### Description

IP cores use the design environment to identify the most appropriate interfaces to connect to the parent system.

Name	Description
NATIVE	Supports native IP interfaces.
QSYS	Supports standard Platform Designer interfaces.





### 9.19.8.3 Direction Properties

Name	Description
BIDIR	Indicates the direction for a bidirectional signal.
INOUT	Indicates the direction for an input signal.
OUTPUT	Indicates the direction for an output signal.



### 9.19.8.4 Element Properties

#### Description

Element properties are, with the exception of `ENABLED` and `NAME`, read-only properties of the types of instances, interfaces, and connections. These read-only properties represent metadata that does not vary between copies of the same type. `ENABLED` and `NAME` properties are specific to particular instances, interfaces, or connections.

Type	Name	Description
String	<code>AUTHOR</code>	The author of the component or interface.
Boolean	<code>AUTO_EXPORT</code>	Indicates whether unconnected interfaces on the instance are automatically exported.
String	<code>CLASS_NAME</code>	The type of the instance, interface or connection, for example, <code>altera_nios2</code> or <code>avalon_slave</code> .
String	<code>DESCRIPTION</code>	The description of the instance, interface or connection type.
String	<code>DISPLAY_NAME</code>	The display name for referencing the type of instance, interface or connection.
Boolean	<code>EDITABLE</code>	Indicates whether you can edit the component in the Platform Designer Component Editor.
Boolean	<code>ENABLED</code>	Indicates whether the instance is enabled.
String	<code>GROUP</code>	The IP Catalog category.
Boolean	<code>INTERNAL</code>	Hides internal IP components or sub-components from the IP Catalog..
String	<code>NAME</code>	The name of the instance, interface or connection.
String	<code>VERSION</code>	The version number of the instance, interface or connection, for example, <code>16.1</code> .



### 9.19.8.5 Instance Properties

Type	Name	Description
String	AUTO_EXPORT	Indicates whether Platform Designer automatically exports the unconnected interfaces on the instance.
Boolean	ENABLED	If true, Platform Designer includes this instance in the generated system.
String	NAME	The name of the system, which Platform Designer uses as the name of the top-level module in the generated HDL.



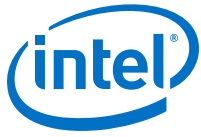
### 9.19.8.6 Interface Properties

Type	Name	Description
String	EXPORT_OF	<p>Indicates which interface of a child instance to export through the top-level interface. Before using this command, you must create the top-level interface using the <code>add_interface</code> command. You must use the format: <code>&lt;instanceName.interfaceName&gt;</code>. For example:</p> <pre>set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port</pre>



### 9.19.8.7 Message Levels Properties

Name	Description
COMPONENT_INFO	Reports an informational message only during component editing.
DEBUG	Provides messages when debug mode is enabled.
ERROR	Provides an error message.
INFO	Provides an informational message.
PROGRESS	Reports progress during generation.
TODOERROR	Provides an error message that indicates the system is incomplete.
WARNING	Provides a warning message.



### 9.19.8.8 Module Properties

Type	Name	Description
String	GENERATION_ID	The generation ID for the system.
String	NAME	The name of the instance.



### 9.19.8.9 Parameter Properties

Type	Name	Description
Boolean	AFFECTS_ELABORATION	Set AFFECTS_ELABORATION to false for parameters that do not affect the external interface of the module. An example of a parameter that does not affect the external interface is <code>isNonVolatileStorage</code> . An example of a parameter that does affect the external interface is <code>width</code> . When the value of a parameter changes and AFFECTS_ELABORATION is false, the elaboration phase does not repeat and improves performance. When AFFECTS_ELABORATION is set to true, the default value, Platform Designer reanalyzes the HDL file to determine the port widths and configuration each time a parameter changes.
Boolean	AFFECTS_GENERATION	The default value of AFFECTS_GENERATION is false if you provide a top-level HDL module. The default value is true if you provide a fileset callback. Set AFFECTS_GENERATION to false if the value of a parameter does not change the results of fileset generation.
Boolean	AFFECTS_VALIDATION	The AFFECTS_VALIDATION property determines whether a parameter's value sets derived parameters, and whether the value affects validation messages. Setting this property to false may improve response time in the parameter editor when the value changes.
String[]	ALLOWED_RANGES	Indicates the range or ranges of the parameter. For integers, each range is a single value, or a range of values defined by a start and end value, and delimited by a colon, for example, <code>11:15</code> . This property also specifies the legal values and description strings for integers, for example, <code>{0:None 1:Monophonic 2:Stereo 4:Quadrophonic}</code> , where 0, 1, 2, and 4 are the legal values. You can assign description strings in the parameter editor for string variables. For example, <pre>ALLOWED_RANGES {"dev1:Cyclone IV GX" "dev2:Stratix V GT" }</pre>
String	DEFAULT_VALUE	The default value.
Boolean	DERIVED	When <code>True</code> , indicates that the parameter value is set by the component and cannot be set by the user. Derived parameters are not saved as part of an instance's parameter values. The default value is <code>False</code> .
String	DESCRIPTION	A short user-visible description of the parameter, suitable for a tooltip description in the parameter editor.
String[]	DISPLAY_HINT	Provides a hint about how to display a property. <ul style="list-style-type: none"> <li><code>boolean</code>--For integer parameters whose value are 0 or 1. The parameter displays as an option that you can turn on or off.</li> <li><code>radio</code>--displays a parameter with a list of values as radio buttons.</li> <li><code>hexadecimal</code>--for integer parameters, displays and interprets the value as a hexadecimal number, for example: <code>0x00000010</code> instead of 16.</li> <li><code>fixed_size</code>--for <code>string_list</code> and <code>integer_list</code> parameters, the <code>fixed_size</code> DISPLAY_HINT eliminates the <b>Add</b> and <b>Remove</b> buttons from tables.</li> </ul>
String	DISPLAY_NAME	The GUI label that appears to the left of this parameter.
String	DISPLAY_UNITS	The GUI label that appears to the right of the parameter.
Boolean	ENABLED	When <code>False</code> , the parameter is disabled. The parameter displays in the parameter editor but is grayed out, indicating that you cannot edit this parameter.
String	GROUP	Controls the layout of parameters in the GUI.



Type	Name	Description
Boolean	HDL_PARAMETER	When <code>True</code> , Platform Designer passes the parameter to the HDL component description. The default value is <code>False</code> .
String	LONG_DESCRIPTION	A user-visible description of the parameter. Similar to <code>DESCRIPTION</code> , but allows a more detailed explanation.
String	NEW_INSTANCE_VALUE	Changes the default value of a parameter without affecting older components that do not explicitly set a parameter value, and use the <code>DEFAULT_VALUE</code> property. Older instances continue to use <code>DEFAULT_VALUE</code> for the parameter and new instances use the value assigned by <code>NEW_INSTANCE_VALUE</code> .
String[]	SYSTEM_INFO	Allows you to assign information about the instantiating system to a parameter that you define. <code>SYSTEM_INFO</code> requires an argument specifying the type of information. For example: <pre>SYSTEM_INFO &lt;info-type&gt;</pre>
String	SYSTEM_INFO_ARG	Defines an argument to pass to <code>SYSTEM_INFO</code> . For example, the name of a reset interface.
(various)	SYSTEM_INFO_TYPE	Specifies the types of system information that you can query. Refer to <i>System Info Type Properties</i> .
(various)	TYPE	Specifies the type of the parameter. Refer to <i>Parameter Type Properties</i> .
(various)	UNITS	Sets the units of the parameter. Refer to <i>Units Properties</i> .
Boolean	VISIBLE	Indicates whether or not to display the parameter in the parameter editor.
String	WIDTH	Indicates the width of the logic vector for the <code>STD_LOGIC_VECTOR</code> parameter.

### Related Links

- [System Info Type Properties](#) on page 589
- [Parameter Type Properties](#) on page 586
- [Units Properties](#) on page 591





### 9.19.8.10 Parameter Status Properties

Type	Name	Description
Boolean	ACTIVE	Indicates that this parameter is an active parameter.
Boolean	DEPRECATED	Indicates that this parameter exists only for backwards compatibility, and may not have any effect.
Boolean	EXPERIMENTAL	Indicates that this parameter is experimental and not exposed in the design flow.



### 9.19.8.11 Parameter Type Properties

Name	Description
BOOLEAN	A boolean parameter set to <code>true</code> or <code>false</code> .
FLOAT	A signed 32-bit floating point parameter. (Not supported for HDL parameters.)
INTEGER	A signed 32-bit integer parameter.
INTEGER_LIST	A parameter that contains a list of 32-bit integers. (Not supported for HDL parameters.)
LONG	A signed 64-bit integer parameter. (Not supported for HDL parameters.)
NATURAL	A 32-bit number that contains values 0 to 2147483647 (0x7fffffff).
POSITIVE	A 32-bit number that contains values 1 to 2147483647 (0x7fffffff).
STD_LOGIC	A single bit parameter set to 0 or 1.
STD_LOGIC_VECTOR	An arbitrary-width number. The parameter property <code>WIDTH</code> determines the size of the logic vector.
STRING	A string parameter.
STRING_LIST	A parameter that contains a list of strings. (Not supported for HDL parameters.)



### 9.19.8.12 Port Properties

Type	Name	Description
(various)	DIRECTION	The direction of the signal. Refer to <i>Direction Properties</i> .
String	ROLE	The type of the signal. Each interface type defines a set of interface types for its ports.
Integer	WIDTH	The width of the signal in bits.

#### Related Links

[Direction Properties](#) on page 577



### 9.19.8.13 Project Properties

Type	Name	Description
String	DEVICE	The device part number in the Intel Quartus Prime project that contains the Platform Designer system.
String	DEVICE_FAMILY	The device family name in the Intel Quartus Prime project that contains the Platform Designer system.



### 9.19.8.14 System Info Type Properties

Type	Name	Description
String	ADDRESS_MAP	An XML-formatted string that describes the address map for the interface specified in the SYSTEM_INFO parameter property.
Integer	ADDRESS_WIDTH	The number of address bits that Platform Designer requires to address memory-mapped slaves connected to the specified memory-mapped master in this instance.
String	AVALON_SPEC	The version of the Platform Designer interconnect. Refer to <i>Avalon Interface Specifications</i> .
Integer	CLOCK_DOMAIN	An integer that represents the clock domain for the interface specified in the SYSTEM_INFO parameter property. If this instance has interfaces on multiple clock domains, you can use this property to determine which interfaces are on each clock domain. The absolute value of the integer is arbitrary.
Long, Integer	CLOCK_RATE	The rate of the clock connected to the clock input specified in the SYSTEM_INFO parameter property. If zero, the clock rate is currently unknown.
String	CLOCK_RESET_INFO	The name of this instance's primary clock or reset sink interface. You use this property to determine the reset sink for global reset when you use SOPC Builder interconnect that conforms to <i>Avalon Interface Specifications</i> .
String	CUSTOM_INSTRUCTION_SLAVES	Provides slave information, including the name, base address, address span, and clock cycle type.
String	DESIGN_ENVIRONMENT	A string that identifies the current design environment. Refer to <i>Design Environment Type Properties</i> .
String	DEVICE	The device part number of the selected device.
String	DEVICE_FAMILY	The family name of the selected device.
String	DEVICE_FEATURES	A list of key/value pairs delimited by spaces that indicate whether a device feature is available in the selected device family. The format of the list is suitable for passing to the array command. The keys are device features. The values are 1 if the feature is present, and 0 if the feature is absent.
String	DEVICE_SPEEDGRADE	The speed grade of the selected device.
Integer	GENERATION_ID	An integer that stores a hash of the generation time that Platform Designer uses as a unique ID for a generation run.
BigInteger, Long	INTERRUPTS_USED	A mask indicating which bits of an interrupt receiver are connected to interrupt senders. The interrupt receiver is specified in the system info argument.
Integer	MAX_SLAVE_DATA_WIDTH	The data width of the widest slave connected to the specified memory-mapped master.
String, Boolean, Integer	QUARTUS_INI	The value of the quartus.ini setting specified in the system info argument.
Integer	RESET_DOMAIN	An integer representing the reset domain for the interface specified in the SYSTEM_INFO parameter property. If this instance has interfaces on multiple reset



Type	Name	Description
		domains, you can use this property to determine which interfaces are on each reset domain. The absolute value of the integer is arbitrary.
String	TRISTATECONDUIT_INFO	An XML description of the tri-state conduit masters connected to a tri-state conduit slave. The slave is specified as the SYSTEM_INFO parameter property. The value contains information about the slave, connected master instance and interface names, and signal names, directions, and widths.
String	TRISTATECONDUIT_MASTERS	The names of the instance's interfaces that are tri-state conduit slaves.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

### Related Links

- [Design Environment Type Properties](#) on page 576
- [Avalon Interface Specifications](#)
- [Platform Designer Interconnect](#) on page 659



### 9.19.8.15 Units Properties

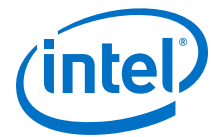
Name	Description
ADDRESS	A memory-mapped address.
BITS	Memory size in bits.
BITSPERSECOND	Rate in bits per second.
BYTES	Memory size in bytes.
CYCLES	A latency or count in clock cycles.
GIGABITSPERSECOND	Rate in gigabits per second.
GIGABYTES	Memory size in gigabytes.
GIGAHERTZ	Frequency in GHz.
HERTZ	Frequency in Hz.
KILOBITSPERSECOND	Rate in kilobits per second.
KILOBYTES	Memory size in kilobytes.
KILOHERTZ	Frequency in kHz.
MEGABITSPERSECOND	Rate, in megabits per second.
MEGABYTES	Memory size in megabytes.
MEGAHERTZ	Frequency in MHz.
MICROSECONDS	Time in microseconds.
MILLISECONDS	Time in milliseconds.
NANOSECONDS	Time in nanoseconds.
NONE	Unspecified units.
PERCENT	A percentage.
PICOSECONDS	Time in picoseconds.
SECONDS	Time in seconds.



### 9.19.8.16 Validation Properties

Type	Name	Description
Boolean	AUTOMATIC_VALIDATION	When <code>true</code> , Platform Designer runs system validation and elaboration after each scripting command. When <code>false</code> , Platform Designer runs system validation with validation scripting commands. Some queries affected by system elaboration may be incorrect if automatic validation is disabled. You can disable validation to make a system script run faster.





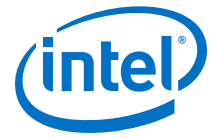
### 9.19.8.17 Interface Direction

Type	Name	Description
String	INPUT	Indicates that the interface is a slave (input, transmitter, sink, or end).
String	OUTPUT	Indicates that the interface is a master (output, receiver, source, or start).



### 9.19.8.18 File Set Kind

Name	Description
EXAMPLE_DESIGN	This file-set contains example design files.
QUARTUS_SYNTH	This file-set contains files that Platform Designer uses for Intel Quartus Prime Synthesis
SIM_VERILOG	This file-set contains files that Platform Designer uses for Verilog HDL Simulation.
SIM_VHDL	This file-set contains files that Platform Designer uses for VHDL Simulation.



### 9.19.8.19 Access Type

Name	Type	Description
String	READ_ONLY	Indicates that the parameter can be only read-only.
String	WRITABLE	Indicates that the parameter has read/write properties.



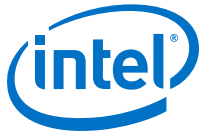
### 9.19.8.20 Instantiation Hdl File Properties

Name	Type	Description
Boolean	CONTAINS_INLINE_CONFIGURATION	Returns <i>True</i> if the HDL file contains inline configuration.
Boolean	IS_CONFIGURATION_PACKAGE	Returns <i>True</i> if the HDL file is a configuration package.
Boolean	IS_TOP_LEVEL	Returns <i>True</i> if the HDL file is the top-level HDL file.
String	OUTPUT_PATH	Specifies the output path of the HDL file.
String	TYPE	Specifies the HDL file type of the HDL file.



### 9.19.8.21 Instantiation Interface Duplicate Type

Type	Name	Description
String	CLONE	Creates a copy of an interface and all the interface ports.
String	MIRROR	Creates a copy of an interface with all the port roles and directions reversed.



### 9.19.8.22 Instantiation Interface Properties

Name	Type	Description
String	DIRECTION	The direction of the interface.
String	TYPE	The type of the interface.



### 9.19.8.23 Instantiation Properties

Name	Type	Description
String	HDL_COMPILATION_LIBRARY	Indicates the HDL compilation library name of the generic component.
String	HDL_ENTITY_NAME	Indicates the HDL entity name of the Generic Component.
String	IP_FILE	Indicates the .ip file path that implements the generic component.



### 9.19.8.24 Port Properties

Name	Type	Description
String	DIRECTION	Specifies the direction of the signal
String	NAME	Renames a top-level port. Only use with <code>set_interface_port_property</code>
String	ROLE	Specifies the type of the signal. Each interface type defines a set of interface types for its ports.
String	VHDL_TYPE	Specifies the VHDL type of the signal. Can be either <code>STANDARD_LOGIC</code> , or <code>STANDARD_LOGIC_VECTOR</code> .
Integer	WIDTH	Specifies the width of the signal in bits.

#### Related Links

[Direction Properties](#) on page 577





### 9.19.8.25 VHDL Type

Name	Description
STD_LOGIC	Represents the value of a digital signal in a wire.
STD_LOGIC_VECTOR	Represents an array of digital signals and variables.

### 9.19.9 Parameterizing an Instantiated IP Core after save\_system Command

When you call the `save_system` command in your Tcl script, Platform Designer converts all the instantiated IP cores in your system to generic components.

To modify these IP cores after saving your system, you must first load the actual component within the instantiated generic component. Re-parameterize an instantiated IP core using one of the following methods:

1. Load the component in the Platform Designer system, modify the component's parameter value, and save the component:

```
...
save_system kernel_system.qsys
...
load_component cra_root
set_component_parameter_value DATA_W 64
save_component
...
```

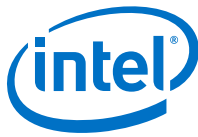
2. Load the `.ip` file specific to the component, modify the instance's parameter value, and save the `.ip` file:

```
...
save_system kernel_system.qsys
...
load_system cra_root.ip
set_instance_parameter_value cra_root DATA_W 64
save_system
...
```

**Note:** To directly modify an instance parameter value after the `save_system` command, you must load the `.ip` file corresponding to the IP component.

#### Related Links

- [set\\_component\\_parameter\\_value](#) on page 521
- [load\\_component](#) on page 518
- [save\\_component](#) on page 520
- [save\\_system](#) on page 413



### 9.19.10 Validate the Generic Components in a System with `qsys-validate`

Use the `qsys-validate` utility to run IP component footprint validation on the `.qsys` file for the system.

**Table 116. `qsys-validate` Command-Line Options**

Option	Usage	Description
<i>1st arg file</i>	Optional	The name of the <b>.qsys</b> system file to validate.
<code>--search-path[=&lt;value&gt;]</code>	Optional	If omitted, Platform Designer uses a standard default path. If provided, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example: <code>/extra/dir.\$</code> .
<code>--strict</code>	Optional	Enables strict validation. All warnings are reported as errors
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	The maximum memory size Platform Designer uses for allocations when running <code>qsys-edit</code> . You specify this value as <code>&lt;size&gt;&lt;unit&gt;</code> , where unit is <code>m</code> (or <code>M</code> ) for multiples of megabytes, or <code>g</code> (or <code>G</code> ) for multiples of gigabytes. The default value is 512m.
<code>--help</code>	Optional	Display help for <code>qsys-validate</code> .

### 9.19.11 Archive a Platform Designer System with `qsys-archive`

The `qsys-archive` command allows you to archive a system, extract an archived system, and retrieve information about the system's dependencies.

**Table 117. `qsys-archive` Command-Line Options**

Option	Usage	Description
<i>&lt;1st arg file&gt;</i>	Required	The filename of the root Platform Designer system, Platform Designer file archive, or the Intel Quartus Prime project file.
<code>--search-path[=&lt;value&gt;]</code>	Optional	If you omit this option, Platform Designer uses a standard default path. If you specify this option, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example: <code>/extra/dir,\$</code> .
<code>--archive</code>	Optional	Creates a zip archive of the specified Platform Designer system or the Intel Quartus Prime project.
<code>--report-file[=&lt;value&gt;]</code>	Optional	Lists the files that the Platform Designer system or the Intel Quartus Prime project references, and writes the files list to the specified name in <code>.txt</code> format.
<code>--output-directory[=&lt;file&gt;]</code>	Optional	Specifies the output directory to save the archive.
<code>--extract</code>	Optional	Extracts all the files in the given archive.
<code>--output-name[=&lt;value&gt;]</code>	Optional	Specifies the output name to save the archive or report.
<code>collect-to-common-directory[=&lt;true/false&gt;]</code>	Optional	When archiving, collects all the <code>.qsys</code> files in the root directory of the archive and all <code>.ip</code> files in a single <code>ip</code> directory, and updates all the matching references. The default option is <code>true</code> .
<i>continued...</i>		



Option	Usage	Description
<code>new-quartus-project [=&lt;value&gt;]</code>	Optional	Creates a new Intel Quartus Prime project which contains all the <code>.ip</code> and system files referenced by the Platform Designer system or the Intel Quartus Prime project.
<code>quartus-project [=&lt;value&gt;]</code>	Optional	When you use this command in combination with: <ul style="list-style-type: none"> <li><code>--report-file</code>—adds all the referenced files to the Intel Quartus Prime project.</li> <li><code>--extract</code>—adds all extracted files to the specified project.</li> <li><code>--archive</code>—archives all the system and <code>.ip</code> files referenced in the Intel Quartus Prime project.</li> </ul>
<code>--rev</code>	Optional	Specifies the name of the Intel Quartus Prime project revision.
<code>--include-generated-files</code>	Optional	Includes all the generated files of the Platform Designer system.
<code>--force</code>	Optional	Forcefully creates the specified archive or report, overwriting any existing archives or reports.
<code>--jvm-max-heap-size=&lt;value&gt;</code>	Optional	Specifies the maximum memory size Platform Designer uses for allocations when running <code>qsys-edit</code> . Specify this value as <code>&lt;size&gt;&lt;unit&gt;</code> , where unit is <code>m</code> (or <code>M</code> ) for multiples of megabytes, or <code>g</code> (or <code>G</code> ) for multiples of gigabytes. The default value is 512m.
<code>--help</code>	Optional	Displays help for <code>qsys-archive</code> .

Alternatively, you can archive and restore your system using the Platform Designer GUI. For more information, refer to *Archive your System* section.

#### Related Links

[Archive your System](#) on page 355

### 9.19.12 Generate an IP Component or Platform Designer System with `quartus_ipgenerate`

The `quartus_ipgenerate` command allows you to generate IP components or a Platform Designer system in your Intel Quartus Prime project. Ensure that you include the IP component or the Platform Designer system you wish to generate in your Intel Quartus Prime project.

To run the `quartus_ipgenerate` command from the Intel Quartus Prime shell, type:

```
quartus_ipgenerate <project name> [<options>]
```

Use any of the following options with the `quartus_ipgenerate` utility:



**Table 118. quartus\_ipgenerate Command-Line Options**

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the Intel Quartus Prime project file (.qpf). This option generates all the .qsys and .ip files in the specified Intel Quartus Prime project (<project name>).
-f [<argument file>]	Optional	Specifies a file containing additional command-line arguments. Arguments that you specify after this option can conflict or override the options you specify in the argument file.
--rev[=<revision name>] or -c[=<revision name>]	Optional	Specifies the Intel Quartus Prime project revision and the associated .qsf file to use. If you omit this option, Platform Designer uses the same revision name as your Intel Quartus Prime project.
--clear_ip_generation_dirs or --clean	Optional	Clears the generation directories of all the .qsys or the .ip files in the specified Intel Quartus Prime project. For example, to clear the generation directories in the project test, run the following command:  <pre>quartus_ipgenerate --clear_ip_generation_dirs test</pre> or  <pre>quartus_ipgenerate --clean test</pre>
--generate_ip_file -- ip_file[=<ip file name>]	Optional	Generates the files for <file name> .ip file in the specified Intel Quartus Prime project. Use the following optional flags with --generate_ip_file: <ul style="list-style-type: none"> <li>-synthesis[=&lt;value&gt;]—optional argument that specifies the synthesis target type. Specify the value as either <i>verilog</i> or <i>vhdl</i>. The default value is <i>verilog</i>.</li> <li>-simulation[=&lt;value&gt;]—optional argument that specifies the simulation target type. Specify the value as either <i>verilog</i> or <i>vhdl</i>. If you omit this flag, Platform Designer does not generate any simulation files.</li> <li>--clear_ip_generation_dirs—clears the preexisting generation directories before generation. If you omit this command, Platform Designer does not clear the generation directories.</li> </ul> For example, to generate the files for a test.qsys file within the project, test: <pre>quartus_ipgenerate --generate_ip_file --synthesis=vhdl --simulation=verilog --clear_ip_generation_dirs --ip_file=test.qsys test</pre>
--generate_project_ip_files [<project name>]	Optional	Generates the files for all the .qsys and .ip files in the specified Intel Quartus Prime project. Use any of the following optional flags with --generate_project_ip_files: <ul style="list-style-type: none"> <li>-synthesis[=&lt;value&gt;]—optional argument that specifies the synthesis target type. Specify the value as either <i>verilog</i> or <i>vhdl</i>. The default value is <i>verilog</i>.</li> <li>-simulation[=&lt;value&gt;]—optional argument that specifies the simulation target type. Specify the value as either <i>verilog</i> or <i>vhdl</i>. If you omit this flag, Platform Designer does not generate any simulation files.</li> <li>--clear_ip_generation_dirs—clears the preexisting generation directories before generation. If you omit this command, Platform Designer does not clear the generation directories.</li> </ul>

*continued...*



Option	Usage	Description
		For example, to generate all the .qsys and .ip files within the project, test: <pre>quartus_ipgenerate --generate_project_ip_files -- synthesis=vhdl --simulation=verilog -- clear_ip_generation_dirs test</pre>
--get_project_ip_files	Optional	Returns a list of the .qsys or .ip files in the specified Intel Quartus Prime project. This option displays each file in a separate Intel Quartus Prime message line. For example, to get a list of .qsys files in the project test, and revision rev: <pre>quartus_ipgenerate --get_project_ip_files test -c rev</pre>
--lower_priority	Optional	Allows you to lower the priority of the current process. This option is useful if you use a single-processor computer, allowing you to use other applications more easily while the Intel Quartus Prime software runs the command in the background.

### 9.19.13 Generate an IP Variation File with ip-deploy

Use the ip-deploy utility to generate an IP variation file (.ip file) in the specified location.

**Table 119. ip-deploy Command-Line Options**

Option	Usage	Description
--component-name[=<value>]	Required	The name of a component you instantiate.
--output-name[=<value>]	Optional	Name for the resulting component; defaults to the component's type name.
--component-parameter[=<value>]	Optional	Repeatable. A single value assignment, like --component-param=WIDTH=11. To assign multiple parameters, use this option several times.
--preset[=<value>]	Optional	Repeatable. The name of a saved preset to use in creating a variation of the IP component. Presets are additive and repeatable.
--family[=<value>]	Optional	Sets the device family
--part[=<value>]	Optional	Sets the device part number. You can also use this command to set the base device, device speed-grade, device family, and device feature's system information.
--output-directory[=<value>]	Optional	This directory contains the output IP variation file. Platform Designer automatically creates the directory if the directory does not exist. If you don't specify an output directory, the output directory is the current working directory.
--search-path[=<value>]	Optional	If you do not specify the search path, the command uses a standard default path. If you provide a search path, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", like /extra/dir,\$.
--jvm-max-heap-size[=<value>]	Optional	The maximum memory size Platform Designer uses for allocations when running qsys-edit. You specify this value as <size><unit>, where unit is m (or M) for multiples of megabytes, or g (or G) for multiples of gigabytes. The default value is 512m.
--help	Optional	Displays help for ip-deploy



## 9.20 Document Revision History

The table below indicates edits made to the *Creating a System With Platform Designer* content since its creation.

**Table 120. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>Changed instances of <i>Qsys Pro</i> to Platform Designer</li> </ul>
2017.05.06	17.0.0	<ul style="list-style-type: none"> <li>Updated the topic - <i>Create/Open Project in Qsys Pro</i></li> <li>Updated the topic - <i>Modify the Target Device</i></li> <li>Updated the topic - <i>Modify the IP Search Path</i></li> <li>Added new topic - <i>Save your System</i></li> <li>Added new topic - <i>Archive your System</i></li> <li>Added new topic - <i>Synchronize IP File References</i></li> <li>Updated the topic - <i>Upgrade Outdated IP Components in Qsys Pro.</i></li> <li>Added new topic - <i>Run System Scripts</i></li> <li>Added new topic - <i>View Avalon Memory Mapped Domains in Your Qsys Pro System</i></li> <li>Updated the topic - <i>Qsys Pro Scripting Command Reference</i> for new Tcl scripting commands</li> <li>Updated the topic - <i>Qsys Pro Scripting Property Reference</i> for new Tcl scripting property</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Implemented Intel rebranding.</li> <li>Implemented Qsys rebranding.</li> <li>Integrated Qsys Pro chapter with Qsys.</li> <li>Added command-line options for qsys-archive.</li> <li>Added command-line options for quartus_ipgenerate.</li> <li>Updated the Qsys Pro scripting commands.</li> <li>Added topic on Qsys Pro design conversion.</li> </ul>
2016.05.03	16.0.0	<ul style="list-style-type: none"> <li>Qsys Command-Line Utilities updated with latest supported command-line options.</li> <li>Added: <i>Generate Header Files</i></li> </ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>Added: <i>Troubleshooting IP or Qsys Pro System Upgrade.</i></li> <li>Added: <i>Generating Version-Agnostic IP and Qsys Pro Simulation Scripts.</i></li> <li>Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime.</i></li> </ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>Changed instances of <i>Quartus II</i> to <i>Quartus Prime.</i></li> </ul>
2015.05.04	15.0.0	<ul style="list-style-type: none"> <li>New figure: <i>Avalon-MM Write Master Timing Waveforms in the Parameters Tab.</i></li> <li>Added <b>Enable ECC protection</b> option, <i>Specify Qsys Interconnect Requirements.</i></li> <li>Added External Memory Interface Debug Toolkit note, <i>Generate a Qsys System.</i></li> <li>Modelsim-Altera now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation, <i>Generating Files for Synthesis and Simulation.</i></li> </ul>
December 2014	14.1.0	<ul style="list-style-type: none"> <li>Create and Manage Hierarchical Qsys Systems.</li> <li>Schematic tab.</li> <li>View and Filter Clock and Reset Domains.</li> <li><b>File &gt; Recent Projects</b> menu item.</li> <li>Updated example: Hierarchical System Using Instance Parameters</li> </ul>
<i>continued...</i>		

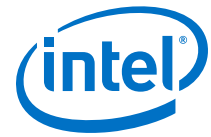


Date	Version	Changes
August 2014	14.0a10.0	<ul style="list-style-type: none"> <li>Added distinction between legacy and standard device generation.</li> <li>Updated: <i>Upgrading Outdated IP Components</i>.</li> <li>Updated: <i>Generating a Qsys System</i>.</li> <li>Updated: <i>Integrating a Qsys System with the Quartus II Software</i>.</li> <li>Added screen shot: <i>Displaying Your Qsys System</i>.</li> </ul>
June 2014	14.0.0	<ul style="list-style-type: none"> <li>Added tab descriptions: Details, Connections.</li> <li>Added <i>Managing IP Settings in the Quartus II Software</i>.</li> <li>Added <i>Upgrading Outdated IP Components</i>.</li> <li>Added <i>Support for Avalon-MM Non-Power of Two Data Widths</i>.</li> </ul>
November 2013	13.1.0	<ul style="list-style-type: none"> <li>Added <i>Integrating with the .qsys File</i>.</li> <li>Added <i>Using the Hierarchy Tab</i>.</li> <li>Added <i>Managing Interconnect Requirements</i>.</li> <li>Added <i>Viewing Qsys Interconnect</i>.</li> </ul>
May 2013	13.0.0	<ul style="list-style-type: none"> <li>Added AMBA APB support.</li> <li>Added qsys-generate utility.</li> <li>Added VHDL BFM ID support.</li> <li>Added <i>Creating Secure Systems (TrustZones)</i>.</li> <li>Added <i>CMSIS Support for Qsys Systems With An HPS Component</i>.</li> <li>Added VHDL language support options.</li> </ul>
November 2012	12.1.0	<ul style="list-style-type: none"> <li>Added AMBA AXI4 support.</li> </ul>
June 2012	12.0.0	<ul style="list-style-type: none"> <li>Added AMBA AX3I support.</li> <li>Added Preset Editor updates.</li> <li>Added command-line utilities, and scripts.</li> </ul>
November 2011	11.1.0	<ul style="list-style-type: none"> <li>Added Synopsys VCS and VCS MX Simulation Shell Script.</li> <li>Added Cadence Incisive Enterprise (NCSIM) Simulation Shell Script.</li> <li>Added <i>Using Instance Parameters and Example Hierarchical System Using Parameters</i>.</li> </ul>
May 2011	11.0.0	<ul style="list-style-type: none"> <li>Added simulation support in Verilog HDL and VHDL.</li> <li>Added testbench generation support.</li> <li>Updated simulation and file generation sections.</li> </ul>
December 2010	10.1.0	Initial release.

### Related Links

#### Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 10 Creating Platform Designer Components

---

You can create a Hardware Component Definition File (`_hw.tcl`) to describe and package IP components for use in a Platform Designer system.

*Note:* Intel now refers to Qsys Pro as Platform Designer.

A `_hw.tcl` describes IP components, interfaces and HDL files. Platform Designer provides the Component Editor to help you create a simple `_hw.tcl` file.

The **Demo AXI Memory** example on the **Platform Designer Design Examples** page of the Altera® web site provides the full code examples that appear in the following topics.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

### Related Links

- [Avalon Interface Specifications](#)
- [Protocol Specifications](#)
- [Demo AXI Memory Example](#)

### 10.1 Platform Designer Components

A Platform Designer component includes the following elements:

- Information about the component type, such as name, version, and author.
- HDL description of the component's hardware, including SystemVerilog, Verilog HDL, or VHDL files.
- A Synopsys Design Constraints File `.sdc` that defines the component for synthesis and simulation.
- A `.ip` file that defines the component's parameters.
- A component's interfaces, including I/O signals.

#### 10.1.1 Interface Support in Platform Designer

IP components (IP Cores) can have any number of interfaces in any combination. Each interface represents a set of signals that you can connect within a Platform Designer system, or export outside of a Platform Designer system.





Platform Designer IP components can include the following interface types:

**Table 121. IP Component Interface Types**

Interface Type	Description
Memory-Mapped	Connects memory-referencing master devices with slave memory devices. Master devices may be processors and DMAs, while slave memory devices may be RAMs, ROMs, and control registers. Data transfers between master and slave may be uni-directional (read only or write only), or bi-directional (read and write).
Streaming	Connects Avalon Streaming (Avalon-ST) sources and sinks that stream unidirectional data, as well as high-bandwidth, low-latency IP components. Streaming creates datapaths for unidirectional traffic, including multichannel streams, packets, and DSP data. The Avalon-ST interconnect is flexible and can implement on-chip interfaces for industry standard telecommunications and data communications cores, such as Ethernet, Interlaken, and video. You can define bus widths, packets, and error conditions.
Interrupts	Connects interrupt senders to interrupt receivers. Platform Designer supports individual, single-bit interrupt requests (IRQs). In the event that multiple senders assert their IRQs simultaneously, the receiver logic (typically under software control) determines which IRQ has highest priority, then responds appropriately
Clocks	Connects clock output interfaces with clock input interfaces. Clock outputs can fan-out without the use of a bridge. A bridge is required only when a clock from an external (exported) source connects internally to more than one source.
Resets	Connects reset sources with reset input interfaces. If your system requires a particular positive-edge or negative-edge synchronized reset, Platform Designer inserts a reset controller to create the appropriate reset signal. If you design a system with multiple reset inputs, the reset controller ORs all reset inputs and generates a single reset output.
Conduits	Connects point-to-point conduit interfaces, or represent signals that are exported from the Platform Designer system. Platform Designer uses conduits for component I/O signals that are not part of any supported standard interface. You can connect two conduits directly within a Platform Designer system as a point-to-point connection, or conduit interfaces can be exported and brought to the top-level of the system as top-level system I/O. You can use conduits to connect to external devices, for example external DDR SDRAM memory, and to FPGA logic defined outside of the Platform Designer system.

### 10.1.2 Component Structure

Intel provides components automatically installed with the Intel Quartus Prime software. You can obtain a list of Platform Designer-compliant components provided by third-party IP developers on Altera's **Intellectual Property & Reference Designs** page by typing: **qsys certified** in the **Search** box, and then selecting **IP Core & Reference Designs**. Components are also provided with Intel development kits, which are listed on the **All Development Kits** page.

Every component is defined with a `<component_name>_hw.tcl` file, a text file written in the Tcl scripting language that describes the component to Platform Designer. When you design your own custom component, you can create the `_hw.tcl` file manually, or by using the Platform Designer Component Editor.

The Component Editor simplifies the process of creating `_hw.tcl` files by creating a file that you can edit outside of the Component Editor to add advanced procedures. When you edit a previously saved `_hw.tcl` file, Platform Designer automatically backs up the earlier version as `_hw.tcl~`.

You can move component files into a new directory, such as a network location, so that other users can use the component in their systems. The `_hw.tcl` file contains relative paths to the other files, so if you move an `_hw.tcl` file, you should also move all the HDL and other files associated with it.

There are four component types:

- **Static**—static components always generate the same output, regardless of their parameterization. Components that instantiate static components must have only static children.
- **Generated**—generated component's fileset callback allows an instance of the component to create unique HDL design files based on the instance's parameter values.
- **Composed**—composed components are subsystems constructed from instances of other components. You can use a composition callback to manage the subsystem in a composed component.
- **Generic**—generic components allow instantiation of IP components without an HDL implementation. Generic components enable hierarchical isolation between system interconnect and IP components.

#### Related Links

- [Create a Composed Component or Subsystem](#) on page 638
- [Add Component Instances to a Static or Generated Component](#) on page 640

### 10.1.3 Component File Organization

A typical component uses the following directory structure where the names of the directories are not significant:

`<component_directory>/`

- `<hdl>/`—Contains the component HDL design files, for example `.v`, `.sv`, or `.vhdl` files that contain the top-level module, along with any required constraint files.
- `<component_name>_hw.tcl`—The component description file.
- `<component_name>_sw.tcl`—The software driver configuration file. This file specifies the paths for the `.c` and `.h` files associated with the component, when required.
- `<software>/`—Contains software drivers or libraries related to the component.

*Note:*

Refer to the *Nios II Software Developer's Handbook* for information about writing a device driver or software package suitable for use with the Nios II processor.

#### Related Links

##### [Nios II Software Developer's Handbook](#)

Refer to the "Nios II Software Build Tools" and "Overview of the Hardware Abstraction Layer" chapters.

### 10.1.4 Component Versions

Platform Designer systems support multiple versions of the same component within the same system; you can create and maintain multiple versions of the same component.

If you have multiple `_hw.tcl` files for components with the same NAME module properties and different VERSION module properties, both versions of the component are available.



If multiple versions of the component are available in the IP Catalog, you can add a specific version of a component by right-clicking the component, and then selecting **Add version** <version\_number>.

#### 10.1.4.1 Upgrade IP Components to the Latest Version

When you open a Platform Designer design, if Platform Designer detects IP components that require regeneration, the **Upgrade IP Cores** dialog box appears and allows you to upgrade outdated components.

Components that you must upgrade in order to successfully compile your design appear in red. Status icons indicate whether a component is currently being regenerated, the component is encrypted, or that there is not enough information to determine the status of component. To upgrade a component, in the **Upgrade IP Cores** dialog box, select the component that you want to upgrade, and then click **Upgrade**. The Intel Quartus Prime software maintains a list of all IP components associated with your design on the **Components** tab in the Project Navigator.

##### Related Links

[Upgrade IP Components Dialog Box](#)  
In *Intel Quartus Prime Help*

## 10.2 Design Phases of an IP Component

When you define a component with the Platform Designer Component Editor, or a custom `_hw.tcl` file, you specify the information that Platform Designer requires to instantiate the component in a Platform Designer system and to generate the appropriate output files for synthesis and simulation.

The following phases describe the process when working with components in Platform Designer:

- **Discovery**—During the discovery phase, Platform Designer reads the `_hw.tcl` file to identify information that appears in the IP Catalog, such as the component's name, version, and documentation URLs. Each time you open Platform Designer, the tool searches for the following file types using the default search locations and entries in the **IP Search Path**:
  - `_hw.tcl` files—Each `_hw.tcl` file defines a single component.
  - IP Index (`.ipx`) files—Each `.ipx` file indexes a collection of available components, or a reference to other directories to search.
- **Static Component Definition**—During the static component definition phase, Platform Designer reads the `_hw.tcl` file to identify static parameter declarations, interface properties, interface signals, and HDL files that define the component. At this stage of the life cycle, the component interfaces may be only partially defined.
- **Parameterization**—During the parameterization phase, after an instance of the component is added to a Platform Designer system, the user of the component specifies parameters with the component's parameter editor.
- **Validation**—During the validation phase, Platform Designer validates the values of each instance's parameters against the allowed ranges specified for each parameter. You can use callback procedures that run during the validation phase to provide validation messages. For example, if there are dependencies between parameters where only certain combinations of values are supported, you can report errors for the unsupported values.

- **Elaboration**—During the elaboration phase, Platform Designer queries the component for its interface information. Elaboration is triggered when an instance of a component is added to a system, when its parameters are changed, or when a system property changes. You can use callback procedures that run during the elaboration phase to dynamically control interfaces, signals, and HDL files based on the values of parameters. For example, interfaces defined with static declarations can be enabled or disabled during elaboration. When elaboration is complete, the component's interfaces and design logic must be completely defined.
- **Composition**—During the composition phase, a component can manipulate the instances in the component's subsystem. The `_hw.tcl` file uses a callback procedure to provide parameterization and connectivity of sub-components.
- **Generation**—During the generation phase, Platform Designer generates synthesis or simulation files for each component in the system into the appropriate output directories, as well as any additional files that support associated tools

### 10.3 Create IP Components in the Platform Designer Component Editor

The Platform Designer Component Editor allows you to create and package an IP component. When you use the Component Editor to define a component, Platform Designer writes the information to an `_hw.tcl` file.

The Platform Designer Component Editor allows you to perform the following tasks:

- Specify component's identifying information, such as name, version, author, etc.
- Specify the SystemVerilog, Verilog HDL, VHDL files, and constraint files that define the component for synthesis and simulation.
- Create an HDL template to define a component interfaces, signals, and parameters.
- Set parameters on interfaces and signals that can alter the component's structure or functionality.

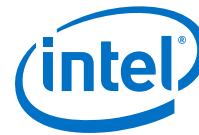
If you add the top-level HDL file that defines the component on **Files** tab in the Platform Designer Component Editor, you must define the component's parameters and signals in the HDL file. You cannot add or remove them in the Component Editor.

If you do not have a top-level HDL component file, you can use the Platform Designer Component Editor to add interfaces, signals, and parameters. In the Component Editor, the order in which the tabs appear reflects the recommended design flow for component development. You can use the **Prev** and **Next** buttons to guide you through the tabs.

In a Platform Designer system, the interfaces of a component are connected in the system, or exported as top-level signals from the system.

If the component is not based on an existing HDL file, enter the parameters, signals, and interfaces first, and then return to the **Files** tab to create the top-level HDL file template. When you click **Finish**, Platform Designer creates the component `_hw.tcl` file with the details that you enter in the Component Editor.

When you save the component, it appears in the IP Catalog.



If you require custom features that the Platform Designer Component Editor does not support, for example, an elaboration callback, use the Component Editor to create the `_hw.tcl` file, and then manually edit the file to complete the component definition.

**Note:** If you add custom coding to a component, do not open the component file in the Platform Designer Component Editor. The Platform Designer Component Editor overwrites your custom edits.

### Example 80. Platform Designer Creates an `_hw.tcl` File from Entries in the Component Editor

```
#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true

add_interface_port clock clk clk Input 1

#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true

add_interface_port reset reset_n reset_n Input 1

#
# connection point streaming
#
add_interface streaming avalon_streaming start
set_interface_property streaming associatedClock clock
set_interface_property streaming associatedReset reset
set_interface_property streaming dataBitsPerSymbol 8
set_interface_property streaming errorDescriptor ""
set_interface_property streaming firstSymbolInHighOrderBits true
set_interface_property streaming maxChannel 0
set_interface_property streaming readyLatency 0
set_interface_property streaming ENABLED true

add_interface_port streaming aso_data data Output 8
add_interface_port streaming aso_valid valid Output 1
add_interface_port streaming aso_ready ready Input 1

#
# connection point slave
#
add_interface slave axi end
set_interface_property slave associatedClock clock
set_interface_property slave associatedReset reset
set_interface_property slave readAcceptanceCapability 1
set_interface_property slave writeAcceptanceCapability 1
set_interface_property slave combinedAcceptanceCapability 1
set_interface_property slave readDataReorderingDepth 1
set_interface_property slave ENABLED true

add_interface_port slave axs_awid awid Input AXI_ID_W
...
add_interface_port slave axs_rresp rresp Output 2
```

### Related Links

[Component Interface Tcl Reference](#) on page 791

### 10.3.1 Save an IP Component and Create the `_hw.tcl` File

You save a component by clicking **Finish** in the Platform Designer Component Editor. The Component Editor saves the component as `<component_name>_hw.tcl` file.

Intel recommends that you move `_hw.tcl` files and their associated files to an `ip/` directory within your Intel Quartus Prime project directory. You can use IP components with other applications, such as the C compiler and a board support package (BSP) generator.

Refer to *Creating a System with Platform Designer* for information on how to search for and add components to the IP Catalog for use in your designs.

#### Related Links

- [Publishing Component Information to Embedded Software \(Nios II Gen 2 Software Developer's Handbook\)](#)
- [Creating a System with Platform Designer](#) on page 327
- [Publishing Component Information to Embedded Software \(Nios II Software Developer's Handbook\)](#)
- [Creating a System with Platform Designer](#) on page 327

### 10.3.2 Edit an IP Component with the Platform Designer Component Editor

In Platform Designer, you make changes to a component by right-clicking the component in the **System Contents** tab, and then clicking **Edit**. After making changes, click **Finish** to save the changes to the `_hw.tcl` file.

You can open an `_hw.tcl` file in a text editor to view the hardware Tcl for the component. If you edit the `_hw.tcl` file to customize the component with advanced features, you cannot use the Component Editor to make further changes without overwriting your customized file.

You cannot use the Component Editor to edit components installed with the Intel Quartus Prime software, such as Intel-provided components. If you edit the HDL for a component and change the interface to the top-level module, you must edit the component to reflect the changes you make to the HDL.

## 10.4 Specify IP Component Type Information

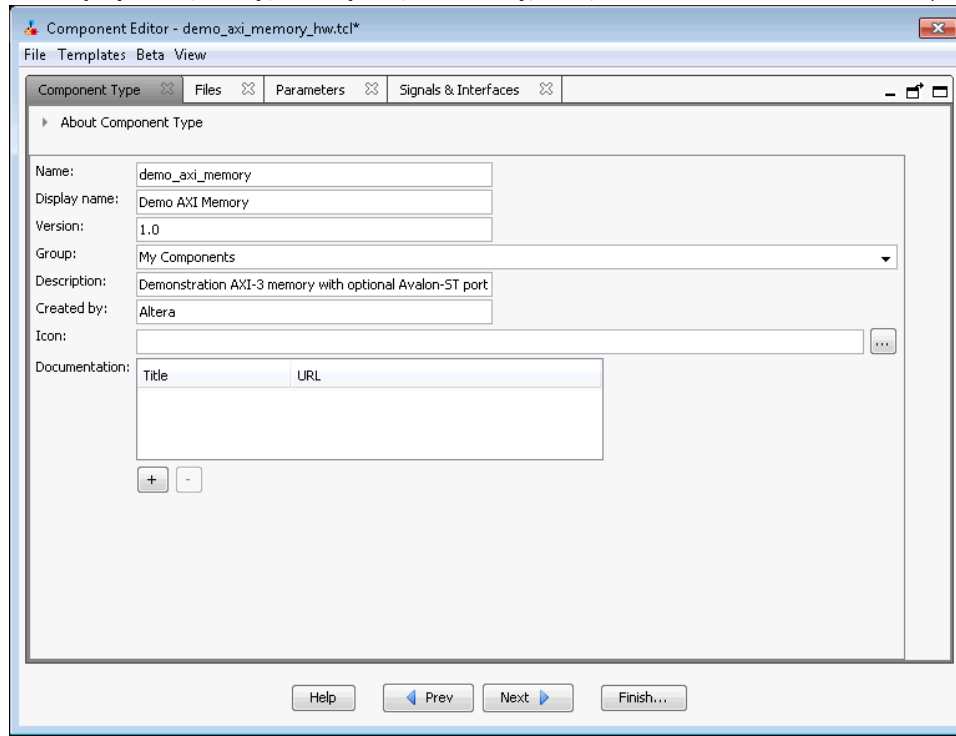
The **Component Type** tab in the Platform Designer Component Editor allows you to specify the following information about the component:



- **Name**—Specifies the name used in the `_hw.tcl` filename, as well as in the top-level module name when you create a synthesis wrapper file for a non HDL-based component.
- **Display name**—Identifies the component in the parameter editor, which you use to configure and instance of the component, and also appears in the IP Catalog under **Project** and on the **System Contents** tab.
- **Version**—Specifies the version number of the component.
- **Group**—Represents the category of the component in the list of available components in the IP Catalog. You can select an existing group from the list, or define a new group by typing a name in the **Group** box. Separating entries in the **Group** box with a slash defines a subcategory. For example, if you type **Memories and Memory Controllers/On-Chip**, the component appears in the IP Catalog under the **On-Chip** group, which is a subcategory of the **Memories and Memory Controllers** group. If you save the component in the project directory, the component appears in the IP Catalog in the group you specified under **Project**. Alternatively, if you save the component in the Intel Quartus Prime installation directory, the component appears in the specified group under **IP Catalog**.
- **Description**—Allows you to describe the component. This description appears when the user views the component details.
- **Created By**—Allows you to specify the author of the component.
- **Icon**—Allows you to enter the relative path to an icon file (`.gif`, `.jpg`, or `.png` format) that represents the component and appears as the header in the parameter editor for the component. The default image is the Intel FPGA IP function icon.
- **Documentation**—Allows you to add links to documentation for the component, and appears when you right-click the component in the IP Catalog, and then select **Details**.
  - To specify an Internet file, begin your path with `http://`, for example:  
`http://mydomain.com/datasheets/my_memory_controller.html`.
  - To specify a file in the file system, begin your path with `file:///` for Linux, and `file://` for Windows; for example (Windows): `file:///company_server/datasheets my_memory_controller.pdf`.

**Figure 188. Component Type Tab in the Component Editor**

The **Display name**, **Group**, **Description**, **Created By**, **Icon**, and **Documentation** entries are optional.



When you use the Component Editor to create a component, it writes this basic component information in the `_hw.tcl` file. The `package require` command specifies the Intel Quartus Prime software version that Platform Designer uses to create the `_hw.tcl` file, and ensures compatibility with this version of the Platform Designer API in future ACDS releases.

**Example 81. `_hw.tcl` Created from Entries in the Component Type Tab**

The component defines its basic information with various module properties using the `set_module_property` command. For example, `set_module_property NAME` specifies the name of the component, while `set_module_property VERSION` allows you to specify the version of the component. When you apply a version to the `_hw.tcl` file, it allows the file to behave exactly the same way in future releases of the Intel Quartus Prime software.

```
# request TCL package from ACDS 14.0
package require -exact qsys 14.0

# demo_axi_memory

set_module_property DESCRIPTION \
"Demo AXI-3 memory with optional Avalon-ST port"

set_module_property NAME demo_axi_memory
set_module_property VERSION 1.0
set_module_property GROUP "My Components"
set_module_property AUTHOR Altera
set_module_property DISPLAY_NAME "Demo AXI Memory"
```





### Related Links

[Component Interface Tcl Reference](#) on page 791

## 10.5 Create an HDL File in the Platform Designer Component Editor

If you do not have an HDL file for your component, you can use the Platform Designer Component Editor to define the component signals, interfaces, and parameters of your component, and then create a simple top-level HDL file.

You can then edit the HDL file to add the logic that describes the component's behavior.

1. In the Platform Designer Component Editor, specify the information about the component in the **Signals & Interfaces**, and **Interfaces**, and **Parameters** tabs.
2. Click the **Files** tab.
3. Click **Create Synthesis File from Signals**.  
The Component Editor creates an HDL file from the specified signals, interfaces, and parameters, and the `.v` file appears in the **Synthesis File** table.

### Related Links

[Specify Synthesis and Simulation Files in the Platform Designer Component Editor](#) on page 618

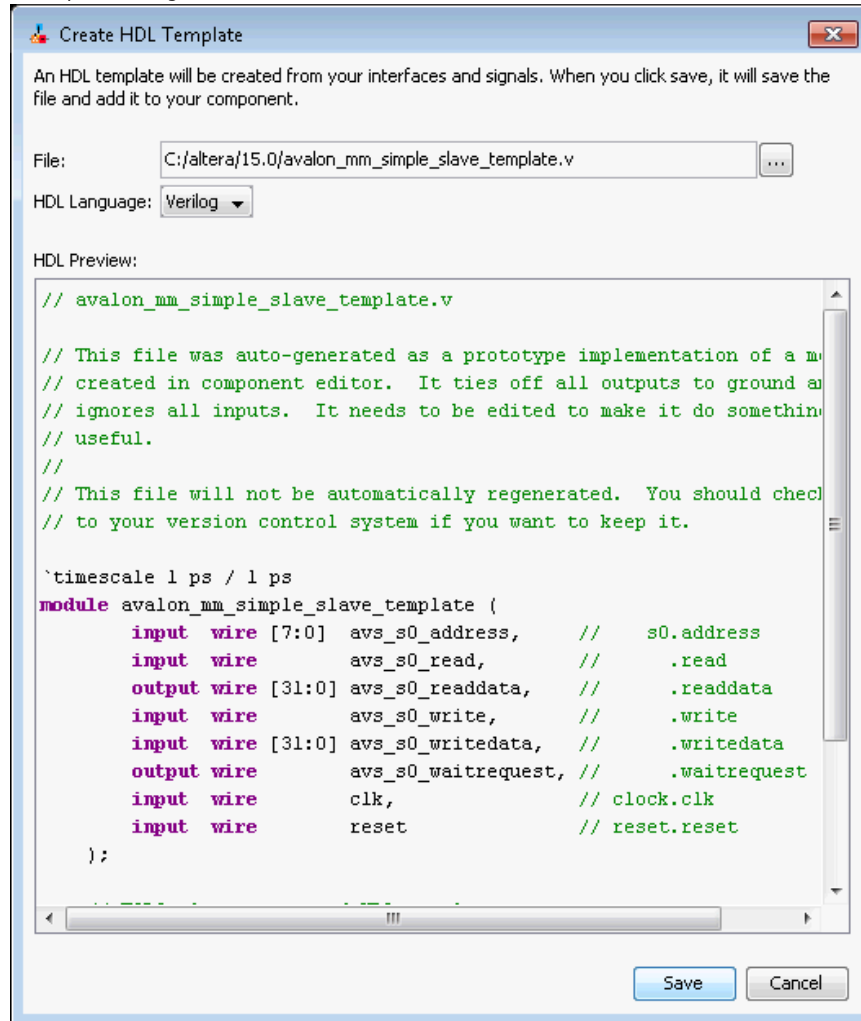
## 10.6 Create an HDL File Using a Template in the Platform Designer Component Editor

You can use a template to create interfaces and signals for your Platform Designer component

1. In Platform Designer, click **New Component** in the IP Catalog.
2. On the **Component Type** tab, define your component information in the **Name**, **Display Name**, **Version**, **Group**, **Description**, **Created by**, **Icon**, and **Documentation** boxes.
3. Click **Finish**.  
Your new component appears in the IP Catalog under the category that you define for "Group".
4. In Platform Designer, right-click your new component in the IP Catalog, and then click **Edit**.
5. In the Platform Designer Component Editor, click any interface from the Templates drop-down menu.  
The Component Editor fills the **Signals** and **Interfaces** tabs with the component interface template details.
6. On the **Files** tab, click **Create Synthesis File from Signals**.
7. Do the following in the **Create HDL Template** dialog box as shown below:
  - a. Verify that the correct files appears in **File** path, or browse to the location where you want to save your file.
  - b. Select the HDL language.

- c. Click **Save** to save your new interface, or **Cancel** to discard the new interface definition.

Create HDL Template Dialog Box



8. Verify the **<component\_name>.v** file appears in the **Synthesis Files** table on the **Files** tab.

### Related Links

[Specify Synthesis and Simulation Files in the Platform Designer Component Editor](#) on page 618

## 10.7 Specify Synthesis and Simulation Files in the Platform Designer Component Editor

The **Files** tab in the Platform Designer Component Editor allows you to specify synthesis and simulation files for your custom component.

If you already have an HDL file that describes the behavior and structure of your component, you can specify those files on the **Files** tab.



If you do not yet have an HDL file, you can specify the signals, interfaces, and parameters of the component in the Component Editor, and then use the **Create Synthesis File from Signals** option on the **Files** tab to create the top-level HDL file. The Component Editor generates the `_hw.tcl` commands to specify the files.

*Note:* After you analyze the component's top-level HDL file (on the **Files** tab), you cannot add or remove signals or change the signal names on the **Signals & Interfaces** tab. If you need to edit signals, edit your HDL source, and then click **Create Synthesis File from Signals** on the **Files** tab to integrate your changes.

A component uses filesets to specify the different sets of files that you can generate for an instance of the component. The supported fileset types are: `QUARTUS_SYNTH`, for synthesis and compilation in the Intel Quartus Prime software, `SIM_VERILOG`, for Verilog HDL simulation, and `SIM_VHDL`, for VHDL simulation.

In an `_hw.tcl` file, you can add a fileset with the `add_fileset` command. You can then list specific files with the `add_fileset_file` command. The `add_fileset_property` command allows you to add properties such as `TOP_LEVEL`.

You can populate a fileset with a fixed list of files, add different files based on a parameter value, or even generate an HDL file with a custom HDL generator function outside of the `_hw.tcl` file.

#### Related Links

- [Create an HDL File in the Platform Designer Component Editor](#) on page 617
- [Create an HDL File Using a Template in the Platform Designer Component Editor](#) on page 617

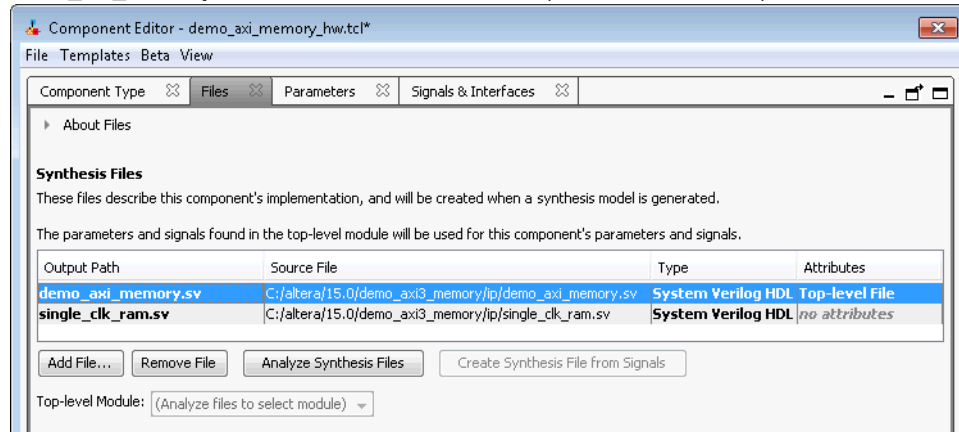
### 10.7.1 Specify HDL Files for Synthesis in the Platform Designer Component Editor

In the Platform Designer Component Editor, you can add HDL files and other support files with options on the `Files` tab.

A component must specify an HDL file as the top-level file. The top-level HDL file contains the top-level module. The **Synthesis Files** list may also include supporting HDL files, such as timing constraints, or other files required to successfully synthesize and compile in the Intel Quartus Prime software. The synthesis files for a component are copied to the generation output directory during Platform Designer system generation.

**Figure 189. Using HDL Files to Define a Component**

In the **Synthesis Files** section on the **Files** tab in the Platform Designer Component Editor, the **demo\_axi\_memory.sv** file should be selected as the top-level file for the component.



## 10.7.2 Analyze Synthesis Files in the Platform Designer Component Editor

After you specify the top-level HDL file in the Platform Designer Component Editor, click **Analyze Synthesis Files** to analyze the parameters and signals in the top-level, and then select the top-level module from the **Top Level Module** list. If there is a single module or entity in the HDL file, Platform Designer automatically populates the **Top-level Module** list.

Once analysis is complete and the top-level module is selected, you can view the parameters and signals on the **Parameters** and **Signals & Interfaces** tabs. The Component Editor may report errors or warnings at this stage, because the signals and interfaces are not yet fully defined.

**Note:** At this stage in the Component Editor flow, you cannot add or remove parameters or signals created from a specified HDL file without editing the HDL file itself.

The synthesis files are added to a fileset with the name `QUARTUS_SYNTH` and type `QUARTUS_SYNTH` in the `_hw.tcl` file created by the Component Editor. The top-level module is used to specify the `TOP_LEVEL` fileset property. Each synthesis file is individually added to the fileset. If the source files are saved in a different directory from the working directory where the `_hw.tcl` is located, you can use standard fixed or relative path notation to identify the file location for the `PATH` variable.

### Example 82. `_hw.tcl` Created from Entries in the Files tab in the Synthesis Files Section

```
# file sets

add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL demo_axi_memory

add_fileset_file demo_axi_memory.sv
SYSTEM_VERILOG PATH demo_axi_memory.sv

add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v
```



### Related Links

- [Specify HDL Files for Synthesis in the Platform Designer Component Editor](#) on page 619
- [Component Interface Tcl Reference](#) on page 791

## 10.7.3 Name HDL Signals for Automatic Interface and Type Recognition in the Platform Designer Component Editor

If you create the component's top-level HDL file before using the Component Editor, the Component Editor recognizes the interface and signal types based on the signal names in the source HDL file. This auto-recognition feature eliminates the task of manually assigning each interface and signal type in the Component Editor.

To enable auto-recognition, you must create signal names using the following naming convention:

*<interface type prefix>\_<interface name>\_<signal type>*

Specifying an interface name with *<interface name>* is optional if you have only one interface of each type in the component definition. For interfaces with only one signal, such as clock and reset inputs, the *<interface type prefix>* is also optional.

**Table 122. Interface Type Prefixes for Automatic Signal Recognition**

When the Component Editor recognizes a valid prefix and signal type for a signal, it automatically assigns an interface and signal type to the signal based on the naming convention. If no interface name is specified for a signal, you can choose an interface name on the **Signals & Interfaces** tab in the Component Editor.

Interface Prefix	Interface Type
asi	Avalon-ST sink (input)
aso	Avalon-ST source (output)
avm	Avalon-MM master
avs	Avalon-MM slave
axm	AXI master
axs	AXI slave
apm	APB master
aps	APB slave
coe	Conduit
csi	Clock Sink (input)
cso	Clock Source (output)
inr	Interrupt receiver
ins	Interrupt sender
ncm	Nios II custom instruction master
ncs	Nios II custom instruction slave
rsi	Reset sink (input)
<i>continued...</i>	

Interface Prefix	Interface Type
rso	Reset source (output)
tcm	Avalon-TC master
tcs	Avalon-TC slave

Refer to the *Avalon Interface Specifications* or the *AMBA Protocol Specification* for the signal types available for each interface type.

**Related Links**

- [Avalon Interface Specifications](#)
- [Protocol Specifications](#)

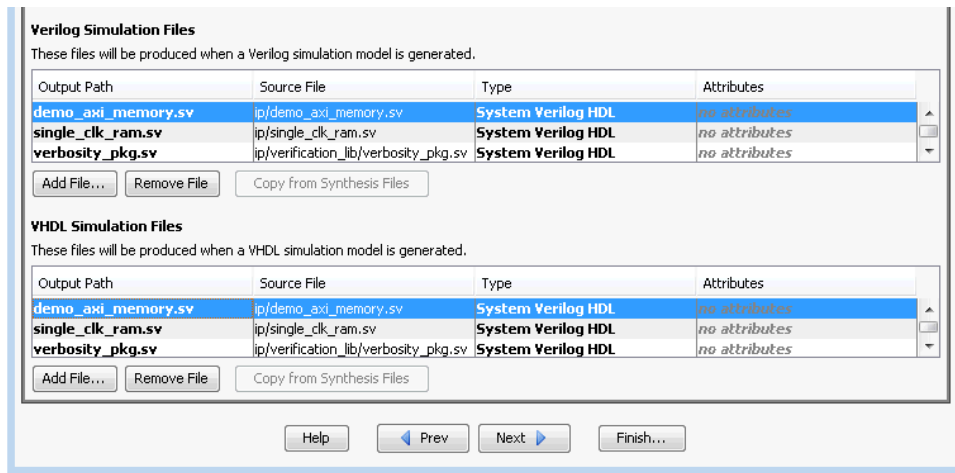
**10.7.4 Specify Files for Simulation in the Component Editor**

To support Platform Designer system generation for your custom component, you must specify VHDL or Verilog simulation files.

You can choose to generate Verilog or VHDL simulation files. In most cases, these files are the same as the synthesis files. If there are simulation-specific HDL files or simulation models, you can use them in addition to, or in place of the synthesis files. To use your synthesis files as your simulation files, click **Copy From Synthesis Files** on the **Files** tab in the Platform Designer Component Editor.

*Note:* The order that you add files to the fileset determines the order of compilation. For VHDL filesets with VHDL files, you must add the files bottom-up, adding the top-level file last.

**Figure 190. Specifying the Simulation Output Files on the Files Tab**



You specify the simulation files in a similar way as the synthesis files with the fileset commands in a `_hw.tcl` file. The code example below shows `SIM_VERILOG` and `SIM_VHDL` filesets for Verilog and VHDL simulation output files. In this example, the same Verilog files are used for both Verilog and VHDL outputs, and there is one additional SystemVerilog file added. This method works for designers of Verilog IP to



support users who want to generate a VHDL top-level simulation file when they have a mixed-language simulation tool and license that can read the Verilog output for the component.

### Example 83. `_hw.tcl` Created from Entries in the Files tab in the Simulation Files Section

```
add_fileset SIM_VERILOG SIM_VERILOG "" ""
set_fileset_property SIM_VERILOG TOP_LEVEL demo_axi_memory
add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv

add_fileset_file demo_axi_memory.sv SYSTEM_VERILOG PATH \
demo_axi_memory.sv

add_fileset SIM_VHDL SIM_VHDL "" ""
set_fileset_property SIM_VHDL TOP_LEVEL demo_axi_memory
set_fileset_property SIM_VHDL ENABLE_RELATIVE_INCLUDE_PATHS false

add_fileset_file demo_axi_memory.sv SYSTEM_VERILOG PATH \
demo_axi_memory.sv

add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv
```

#### Related Links

[Component Interface Tcl Reference](#) on page 791

## 10.7.5 Include an Internal Register Map Description in the .svd for Slave Interfaces Connected to an HPS Component

Platform Designer supports the ability for IP component designers to specify register map information on their slave interfaces. This allows components with slave interfaces that are connected to an HPS component to include their internal register description in the generated .svd file.

To specify their internal register map, the IP component designer must write and generate their own .svd file and attach it to the slave interface using the following command:

```
set_interface_property <slave interface> CMSIS_SVD_FILE <file path>
```

The CMSIS\_SVD\_VARIABLES interface property allows for variable substitution inside the .svd file. You can dynamically modify the character data of the .svd file by using the CMSIS\_SVD\_VARIABLES property.

### Example 84. Setting the CMSIS\_SVD\_VARIABLES Interface Property

For example, if you set the CMSIS\_SVD\_VARIABLES in the `_hw.tcl` file, then in the .svd file if there is a variable `{width}` that describes the element `<size>${width}</size>`, it is replaced by `<size>23</size>` during generation of the .svd file. Note that substitution works only within character data (the data enclosed by `<element>...</element>`) and not on element attributes.

```
set_interface_property <interface name> \
CMSIS_SVD_VARIABLES "{width} {23}"
```



### Related Links

- [Component Interface Tcl Reference](#) on page 791
- [CMSIS - Cortex Microcontroller Software](#)

## 10.8 Add Signals and Interfaces in the Platform Designer Component Editor

In the Platform Designer Component Editor, the **Signals & Interfaces** tab allows you to add signals and interfaces for your custom IP component.

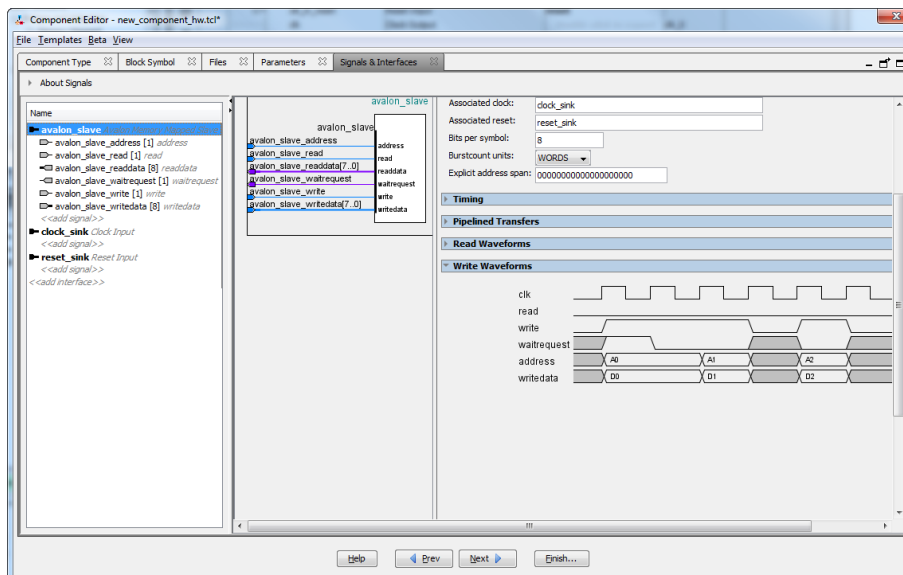
As you select interfaces and associated signals, you can customize the parameters. Messages appear as you add interfaces and signals to guide you when customizing the component. In the parameter editor, a block diagram displays for each interface. Some interfaces display waveforms to show the timing of the interface. If you update timing parameters, the waveforms update automatically.

1. In Platform Designer, click **New Component** in the IP Catalog.
2. In the Platform Designer Component Editor, click the **Signals & Interfaces** tab.
3. To add an interface, click `<<add interface>>` in the left pane. A drop-down list appears where you select the interface type.
4. Select an interface from the drop-down list. The selected interface appears in the parameter editor where you can specify its parameters.
5. To add signals for the selected interface click `<<add signal>>` below the selected interface.
6. To move signals between interfaces, select the signal, and then drag it to another interface.
7. To rename a signal or interface, select the element, and then press **F2**.
8. To remove a signal or interface, right-click the element, and then click **Remove**. Alternatively, to remove a signal or interface, you can select the element, and then press **Delete**. When you remove an interface, Platform Designer also removes all of its associated signals.





Figure 191. Platform Designer Signals & Interfaces tab



## 10.9 Specify Parameters in the Platform Designer Component Editor

Components can include parameterized HDL, which allow users of the component flexibility in meeting their system requirements. For example, a component may have a configurable memory size or data width, where one HDL implementation can be used in different systems, each with unique parameters values.

The **Parameters** tab allows you specify the parameters that are used to configure instances of the component in a Platform Designer system. You can specify various properties for each parameter that describe how to display and use the parameter. You can also specify a range of allowed values that are checked during the validation phase. The **Parameters** table displays the HDL parameters that are declared in the top-level HDL module. If you have not yet created the top-level HDL file, the parameters that you create on the **Parameters** tab are included in the top-level synthesis file template created from the **Files** tab.

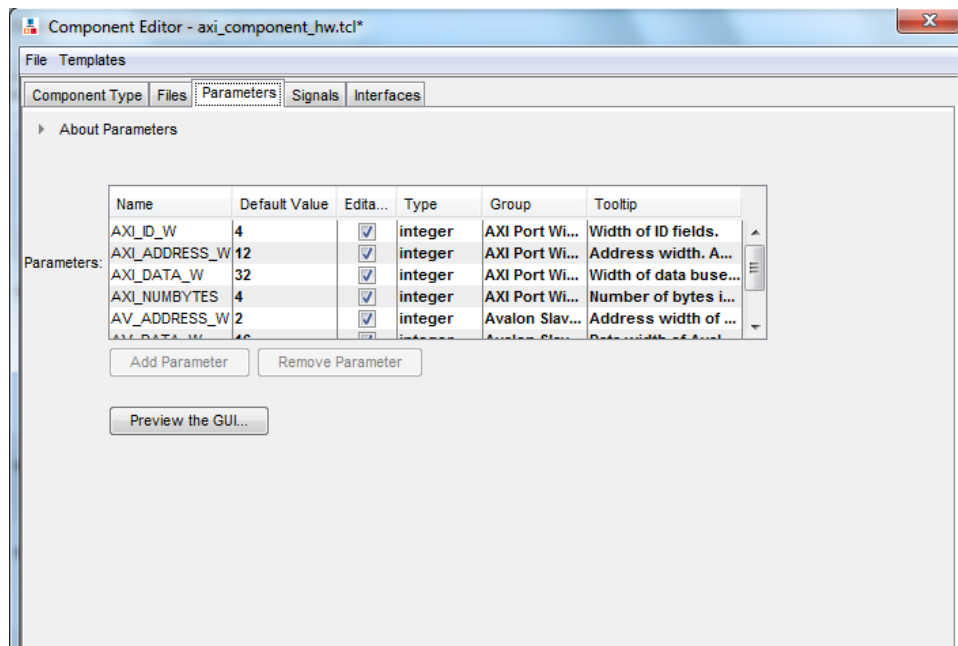
When the component includes HDL files, the parameters match those defined in the top-level module, and you cannot be add or remove them on the **Parameters** tab. To add or remove the parameters, edit your HDL source, and then re-analyze the file.

If you used the Component Editor to create a top-level template HDL file for synthesis, you can remove the newly-created file from the **Synthesis Files** list on the **Files** tab, make your parameter changes, and then re-analyze the top-level synthesis file.

You can use the **Parameters** table to specify the following information about each parameter:

- **Name**—Specifies the name of the parameter.
- **Default Value**—Sets the default value used in new instances of the component.
- **Editable**—Specifies whether or not the user can edit the parameter value.
- **Type**—Defines the parameter type as string, integer, boolean, std\_logic, logic vector, natural, or positive.
- **Group**—Allows you to group parameters in parameter editor.
- **Tooltip**—Allows you to add a description of the parameter that appears when the user of the component points to the parameter in the parameter editor.

**Figure 192. Parameters Tab in the Platform Designer Components Editor**



On the **Parameters** tab, you can click **Preview the GUI** at any time to see how the declared parameters appear in the parameter editor. Parameters with their default values appear with checks in the **Editable** column, indicating that users of this component are allowed to modify the parameter value. Editable parameters cannot contain computed expressions. You can group parameters under a common heading or section in the parameter editor with the **Group** column, and a tooltip helps users of the component understand the function of the parameter. Various parameter properties allow you to customize the component’s parameter editor, such as using radio buttons for parameter selections, or displaying an image.

**Example 85. \_hw.tcl Created from Entries in the Parameters Tab**

In this example, the first `add_parameter` command includes commonly-specified properties. The `set_parameter_property` command specifies each property individually. The **Tooltip** column on the **Parameters** tab maps to the `DESCRIPTION` property, and there is an additional unused `UNITS` property created in the code. The



HDL\_PARAMETER property specifies that the value of the parameter is specified in the HDL instance wrapper when creating instances of the component. The **Group** column in the **Parameters** tab maps to the display items section with the `add_display_item` commands.

**Note:** If a parameter `<n>` defines the width of a signal, the signal width must follow the format: `<n-1>:0`.

```
#
# parameters
#
add_parameter AXI_ID_W INTEGER 4 "Width of ID fields"
set_parameter_property AXI_ID_W DEFAULT_VALUE 4
set_parameter_property AXI_ID_W DISPLAY_NAME AXI_ID_W
set_parameter_property AXI_ID_W TYPE INTEGER
set_parameter_property AXI_ID_W UNITS None
set_parameter_property AXI_ID_W DESCRIPTION "Width of ID fields"
set_parameter_property AXI_ID_W HDL_PARAMETER true
add_parameter AXI_ADDRESS_W INTEGER 12
set_parameter_property AXI_ADDRESS_W DEFAULT_VALUE 12

add_parameter AXI_DATA_W INTEGER 32
...
#
# display items
#
add_display_item "AXI Port Widths" AXI_ID_W PARAMETER ""
```

**Note:** If an AXI slave's ID bit width is smaller than required for your system, the AXI slave response may not reach all AXI masters. The formula of an AXI slave ID bit width is calculated as follows:

$$\text{maximum\_master\_id\_width\_in\_the\_interconnect} + \log_2(\text{number\_of\_masters\_in\_the\_same\_interconnect})$$

For example, if an AXI slave connects to three AXI masters and the maximum AXI master ID length of the three masters is 5 bits, then the AXI slave ID is 7 bits, and is calculated as follows:

$$5 \text{ bits} + 2 \text{ bits } (\log_2(3 \text{ masters})) = 7$$

**Table 123. AXI Master and Slave Parameters**

Platform Designer refers to AXI interface parameters to build AXI interconnect. If these parameter settings are incompatible with the component's HDL behavior, Platform Designer interconnect and transactions may not work correctly. To prevent unexpected interconnect behavior, you must set the AXI component parameters.

AXI Master Parameters	AXI Slave Parameters
readIssuingCapability	readAcceptanceCapability
writeIssuingCapability	writeAcceptanceCapability
combinedIssuingCapability	combinedAcceptanceCapability
	readDataReorderingDepth

**Related Links**

[Component Interface Tcl Reference on page 791](#)

### 10.9.1 Valid Ranges for Parameters in the `_hw.tcl` File

In the `_hw.tcl` file, you can specify valid ranges for parameters.

Platform Designer validation checks each parameter value against the `ALLOWED_RANGES` property. If the values specified are outside of the allowed ranges, Platform Designer displays an error message. Specifying choices for the allowed values enables users of the component to choose the parameter value from a drop-down list or radio button in the parameter editor GUI instead of entering a value.

The `ALLOWED_RANGES` property is a list of valid ranges, where each range is a single value, or a range of values defined by a start and end value.

**Table 124. `ALLOWED_RANGES` Property**

<code>ALLOWED_RANGES</code> Property	Values
<code>{a b c}</code>	a, b, or c
<code>{"No Control" "Single Control" "Dual Controls"}</code>	Unique string values. Quotation marks are required if the strings include spaces .
<code>{1 2 4 8 16}</code>	1, 2, 4, 8, or 16
<code>{1:3}</code>	1 through 3, inclusive.
<code>{1 2 3 7:10}</code>	1, 2, 3, or 7 through 10 inclusive.

**Related Links**

[Declare Parameters with Custom `\_hw.tcl` Commands](#) on page 630

### 10.9.2 Types of Platform Designer Parameters

Platform Designer uses the following parameter types: user parameters, system information parameters, and derived parameters.

[Platform Designer User Parameters](#) on page 628

[Platform Designer System Information Parameters](#) on page 629

[Platform Designer Derived Parameters](#) on page 629

**Related Links**

[Declare Parameters with Custom `\_hw.tcl` Commands](#) on page 630

#### 10.9.2.1 Platform Designer User Parameters

User parameters are parameters that users of a component can control, and appear in the parameter editor for instances of the component. User parameters map directly to parameters in the component HDL. For user parameter code examples, such as `AXI_DATA_W` and `ENABLE_STREAM_OUTPUT`, refer to *Declaring Parameters with Custom `hw.tcl` Commands*.



### 10.9.2.2 Platform Designer System Information Parameters

A `SYSTEM_INFO` parameter is a parameter whose value is set automatically by the Platform Designer system. When you define a `SYSTEM_INFO` parameter, you provide an `information` type, and additional arguments.

For example, you can configure a parameter to store the clock frequency driving a clock input for your component. To do this, define the parameter as `SYSTEM_INFO` of type `CLOCK_RATE`:

```
set_parameter_property <param> SYSTEM_INFO CLOCK_RATE
```

You then set the name of the clock interface as the `SYSTEM_INFO_ARG` argument:

```
set_parameter_property <param> SYSTEM_INFO_ARG <clkname>
```

#### 10.9.2.2.1 Obtaining Device Trait Information Using `PART_TRAIT` System Information Parameter

Within Platform Designer, an IP core can obtain information on the particular traits of a device using the `PART_TRAIT` system info parameter. This system info parameter takes an argument corresponding to the desired part trait. The requested trait must match the trait name as specified in the device database.

*Note:* Using this API declares your core as dependent on the requested trait.

To get the part number setting of Platform Designer system, use the value `DEVICE`, with the `SYSTEM_INFO_ARG` parameter property:

```
add_parameter part_trait_device string ""
set_parameter_property part_trait_device SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_device SYSTEM_INFO_ARG DEVICE
```

To get the base device of the part number setting of Platform Designer system, use the value `BASE_DEVICE`, with the `SYSTEM_INFO_ARG` parameter property:

```
add_parameter part_trait_bd string ""
set_parameter_property part_trait_bd SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_bd SYSTEM_INFO_ARG BASE_DEVICE
```

To get the device speed-grade of the part number setting of Platform Designer system, use the value `DEVICE_SPEEDGRADE`, with the `SYSTEM_INFO_ARG` parameter property:

```
add_parameter part_trait_sg string ""
set_parameter_property part_trait_sg SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_sg SYSTEM_INFO_ARG DEVICE_SPEEDGRADE
```

#### 10.9.2.3 Platform Designer Derived Parameters

Derived parameter values are calculated from other parameters during the Elaboration phase, and are specified in the `hw.tcl` file with the `DERIVED` property. Derived parameter values are calculated from other parameters during the Elaboration phase, and are specified in the `hw.tcl` file with the `DERIVED` property. For example, you can derive a clock period parameter from a data rate parameter. Derived parameters are



sometimes used to perform operations that are difficult to perform in HDL, such as using logarithmic functions to determine the number of address bits that a component requires.

### Related Links

[Declare Parameters with Custom `\_hw.tcl` Commands](#) on page 630

#### 10.9.2.3.1 Parameterized Parameter Widths

Platform Designer allows a `std_logic_vector` parameter to have a width that is defined by another parameter, similar to derived parameters. The width can be a constant or the name of another parameter.

### 10.9.3 Declare Parameters with Custom `_hw.tcl` Commands

The example below illustrates a custom `_hw.tcl` file, with more advanced parameter commands than those generated when you specify parameters in the Component Editor. Commands include the `ALLOWED_RANGES` property to provide a range of values for the `AXI_ADDRESS_W` (**Address Width**) parameter, and a list of parameter values for the `AXI_DATA_W` (**Data Width**) parameter. This example also shows the parameter `AXI_NUMBYTES` (**Data width in bytes**) parameter; that uses the `DERIVED` property. In addition, these commands illustrate the use of the `GROUP` property, which groups some parameters under a heading in the parameter editor GUI. You use the `ENABLE_STREAM_OUTPUT_GROUP` (**Include Avalon streaming source port**) parameter to enable or disable the optional Avalon-ST interface in this design, and is displayed as a check box in the parameter editor GUI because the parameter is of type `BOOLEAN`. Refer to figure below to see the parameter editor GUI resulting from these `hw.tcl` commands.

#### Example 86. Parameter Declaration

In this example, the `AXI_NUMBYTES` parameter is derived during the Elaboration phase based on another parameter, instead of being assigned to a specific value. `AXI_NUMBYTES` describes the number of bytes in a word of data. Platform Designer calculates the `AXI_NUMBYTES` parameter from the `DATA_WIDTH` parameter by dividing by 8. The `_hw.tcl` code defines the `AXI_NUMBYTES` parameter as a derived parameter, since its value is calculated in an elaboration callback procedure. The `AXI_NUMBYTES` parameter value is not editable, because its value is based on another parameter value.

```
add_parameter AXI_ADDRESS_W INTEGER 12

set_parameter_property AXI_ADDRESS_W DISPLAY_NAME \
"AXI Slave Address Width"

set_parameter_property AXI_ADDRESS_W DESCRIPTION \
"Address width."

set_parameter_property AXI_ADDRESS_W UNITS bits
set_parameter_property AXI_ADDRESS_W ALLOWED_RANGES 4:16
set_parameter_property AXI_ADDRESS_W HDL_PARAMETER true

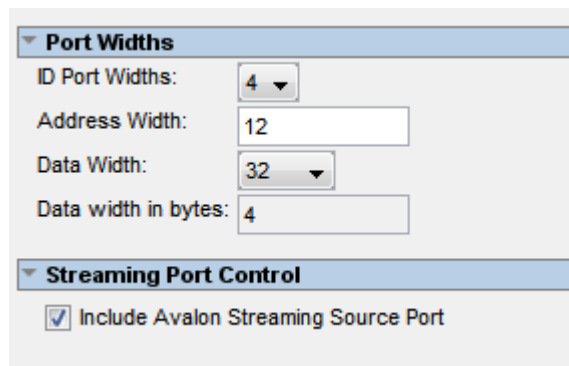
set_parameter_property AXI_ADDRESS_W GROUP \
"AXI Port Widths"

add_parameter AXI_DATA_W INTEGER 32
set_parameter_property AXI_DATA_W DISPLAY_NAME "Data Width"
```



```
set_parameter_property AXI_DATA_W DESCRIPTION \  
"Width of data buses."  
  
set_parameter_property AXI_DATA_W UNITS bits  
  
set_parameter_property AXI_DATA_W ALLOWED_RANGES \  
{8 16 32 64 128 256 512 1024}  
  
set_parameter_property AXI_DATA_W HDL_PARAMETER true  
set_parameter_property AXI_DATA_W GROUP "AXI Port Widths"  
  
add_parameter AXI_NUMBYTES INTEGER 4  
set_parameter_property AXI_NUMBYTES DERIVED true  
  
set_parameter_property AXI_NUMBYTES DISPLAY_NAME \  
"Data Width in bytes; Data Width/8"  
  
set_parameter_property AXI_NUMBYTES DESCRIPTION \  
"Number of bytes in one word"  
  
set_parameter_property AXI_NUMBYTES UNITS bytes  
set_parameter_property AXI_NUMBYTES HDL_PARAMETER true  
set_parameter_property AXI_NUMBYTES GROUP "AXI Port Widths"  
  
add_parameter ENABLE_STREAM_OUTPUT BOOLEAN true  
  
set_parameter_property ENABLE_STREAM_OUTPUT DISPLAY_NAME \  
"Include Avalon Streaming Source Port"  
  
set_parameter_property ENABLE_STREAM_OUTPUT DESCRIPTION \  
"Include optional Avalon-ST source (default),\  
or hide the interface"  
  
set_parameter_property ENABLE_STREAM_OUTPUT GROUP \  
"Streaming Port Control"  
  
...
```

Figure 193. Resulting Parameter Editor GUI from Parameter Declarations



#### Related Links

- [Control Interfaces Dynamically with an Elaboration Callback](#) on page 635
- [Component Interface Tcl Reference](#) on page 791

### 10.9.4 Validate Parameter Values with a Validation Callback

You can use a validation callback procedure to validate parameter values with more complex validation operations than the `ALLOWED_RANGES` property allows. You define a validation callback by setting the `VALIDATION_CALLBACK` module property to the name of the Tcl callback procedure that runs during the validation phase. In the validation callback procedure, the current parameter values is queried, and warnings or errors are reported about the component's configuration.

#### Example 87. Demo AXI Memory Example

If the optional Avalon streaming interface is enabled, then the control registers must be wide enough to hold an AXI RAM address, so the designer can add an error message to ensure that the user enters allowable parameter values.

```
set_module_property VALIDATION_CALLBACK validate
proc validate {} {
  if {
    [get_parameter_value ENABLE_STREAM_OUTPUT ] &&
    ([get_parameter_value AXI_ADDRESS_W] >
     [get_parameter_value AV_DATA_W])
  }
  send_message error "If the optional Avalon streaming port\
is enabled, the AXI Data Width must be equal to or greater\
than the Avalon control port Address Width"
}
}
```

#### Related Links

- [Component Interface Tcl Reference](#) on page 791
- [Demo AXI Memory Example](#)

### 10.10 Declaring SystemVerilog Interfaces in `_hw.tcl`

Platform Designer supports interfaces written in SystemVerilog.

The following example is `_hw.tcl` for a module with a SystemVerilog interface. The sample code is divided into parts 1 and 2.

Part 1 defines the normal array of parameters, Platform Designer interface, and ports

#### Example 88. Example Part 1: Parameters, Platform Designer Interface, and Ports in `_hw.tcl`

```
# request TCL package from ACDS 17.1
#
package require -exact qsys 17.1

#
# module ram_ip_sv_ifc_hw
#
set_module_property DESCRIPTION ""
set_module_property NAME ram_ip_sv_ifc_hw
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME ram_ip_hw_with_SV_d0
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
```





```

set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false

# Part 1 - Add parameter, platform designer interface and ports
# Adding parameter
add_parameter my_interface_parameter STRING "" "I am an interface parameter"

# Adding platform designer interface clk
add_interface clk clock end
set_interface_property clk clockRate 0
# Adding ports to clk interface
add_interface_port clk clk clk Input 1

# Adding platform designer interface reset
add_interface reset reset end
set_interface_property reset associatedClock clk
#Adding ports to reset interface
add_interface_port reset reset reset Input 1

# Adding platform designer interface avalon_slave
add_interface avalon_slave avalon end
set_interface_property avalon_slave addressUnits WORDS
# Adding ports to avalon_slave interface
add_interface_port avalon_slave address address Input 10
add_interface_port avalon_slave write write Input 1
add_interface_port avalon_slave readdata readdata Output 32
add_interface_port avalon_slave writedata writedata Input 32
set_interface_property avalon_slave associatedClock clk
set_interface_property avalon_slave associatedReset reset

#Adding ram_ip files
add_fileset synthesis_fileset QUARTUS_SYNTH
set_fileset_property synthesis_fileset TOP_LEVEL ram_ip
add_fileset_file ram_ip.sv SYSTEM_VERILOG PATH ram_ip.sv

```

Part 2 defines the interface name, ports, and parameters of the SystemVerilog interface.

#### Example 89. Example Part 2: SystemVerilog Interface Parameters in `_hw.tcl`

```

# Part 2 - Adding SV interface and its properties.
# Adding SV interface
add_sv_interface bus mem_ifc

# Setting the parameter property to add SV interface parameters
set_parameter_property my_interface_parameter SV_INTERFACE_PARAMETER bus

# Setting the port properties to add them to SV interface port
set_port_property clk SV_INTERFACE_PORT bus
set_port_property reset SV_INTERFACE_PORT bus

# Setting the port properties to add them as signals inside SV interface
set_port_property address SV_INTERFACE_SIGNAL bus
set_port_property write SV_INTERFACE_SIGNAL bus
set_port_property writedata SV_INTERFACE_SIGNAL bus
set_port_property readdata SV_INTERFACE_SIGNAL bus

#Adding the SV Interface File
add_fileset_file mem_ifc.sv SYSTEM_VERILOG PATH mem_ifc.sv
SYSTEMVERILOG_INTERFACE

```

## 10.11 User Alterable HDL Parameters in `_hw.tcl`

Platform Designer supports the ability to reconfigure features of parameterized modules, such as data bus width or FIFO depth. Platform Designer creates an HDL wrapper when you perform **Generate HDL**. By modifying your `_hw.tcl` files to specify parameter attributes and port properties, you can use Platform Designer to generate reusable RTL.

1. To define an alterable HDL parameter, you must declare the following two attributes for the parameter:
  - `set_parameter_property <parameter_name> HDL_PARAMETER true`
  - `set_parameter_property <parameter_name> AFFECTS_GENERATION false`
2. To have parameterized ports created in the instantiation wrapper, you can either set the width expression when adding a port to an interface, or set the width expression in the port property in `_hw.tcl`:
  - To set the width expression when adding a port:

```
add_interface_port <interface> <port> <signal_type> <direction>
<width_expression>
```
  - To set the width expression in the port property:

```
set_port_property <port> WIDTH_EXPR <width_expression>
```
3. To create and generate the IP component in Platform Designer editor, click the **Open System > IP Variant** tab, specify the new IP variant name in the **IP Variant** field and choose the `_hw.tcl` file that defines user alterable HDL parameters in the **Component type** field.
4. Click **Generate HDL** to generate the IP core. Platform Designer generates a parameterized HDL module for you directly.

To instantiate the IP component in your HDL file, click **Generate > Show Instantiation Template** in the Platform Designer editor to display an instantiation template in Verilog or VHDL. Now you can instantiate the IP core in your top-level design HDL file with the template code.

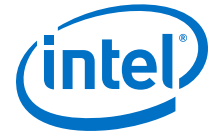
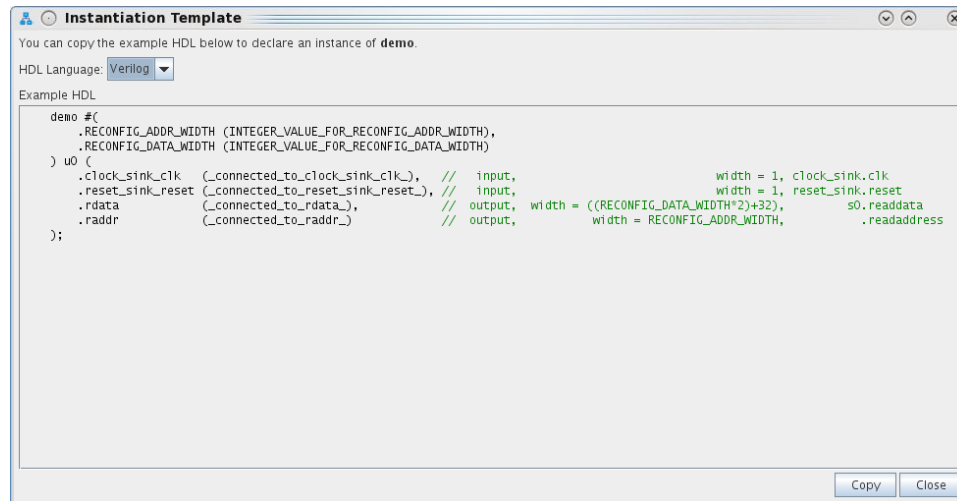


Figure 194. Instantiation Template Dialog Box



The following sample contains `_hw.tcl` to set exportable width values:

### Example 90. Sample `_hw.tcl` Component with User Alterable Expressions

```
package require -exact qsys 17.1

set_module_property NAME demo
set_module_property DISPLAY_NAME "Demo"
set_module_property ELABORATION_CALLBACK elaborate

# add exportable hdl parameter RECONFIG_DATA_WIDTH
add_parameter RECONFIG_DATA_WIDTH INTEGER 48
set_parameter_property RECONFIG_DATA_WIDTH AFFECTS_GENERATION false
set_parameter_property RECONFIG_DATA_WIDTH HDL_PARAMETER true

# add exportable hdl parameter RECONFIG_ADDR_WIDTH
add_parameter RECONFIG_ADDR_WIDTH INTEGER 32
set_parameter_property RECONFIG_ADDR_WIDTH AFFECTS_GENERATION false
set_parameter_property RECONFIG_ADDR_WIDTH HDL_PARAMETER true

# add non-exportable hdl parameter
add_parameter l_addr INTEGER 32
set_parameter l_addr HDL_PARAMETER false

# add interface
add_interface s0 conduit end

proc elaborate {} {
  add_interface_port s0 rdata readdata output "reconfig_data_width*2 +
  l_addr"
  add_interface_port s0 raddr readaddress output [get_parameter_value
  RECONFIG_ADDR_WIDTH]
  set_port_property raddr WIDTH_EXPR "RECONFIG_ADDR_WIDTH"
}
```

## 10.12 Control Interfaces Dynamically with an Elaboration Callback

You can allow user parameters to dynamically control your component's behavior with an elaboration callback procedure during the elaboration phase. Using an elaboration callback allows you to change interface properties, remove interfaces, or add new interfaces as a function of a parameter value. You define an elaboration callback by

setting the module property `ELABORATION_CALLBACK` to the name of the Tcl callback procedure that runs during the elaboration phase. In the callback procedure, you can query the parameter values of the component instance, and then change the interfaces accordingly.

### Example 91. Avalon-ST Source Interface Optionally Included in a Component Specified with an Elaboration Callback

```
set_module_property ELABORATION_CALLBACK elaborate

proc elaborate {} {

    # Optionally disable the Avalon- ST data output

    if{[ get_parameter_value ENABLE_STREAM_OUTPUT] == "false" }{
        set_port_property aso_data      termination true
        set_port_property aso_valid     termination true
        set_port_property aso_ready     termination true
        set_port_property aso_ready     termination_value 0
    }

    # Calculate the Data Bus Width in bytes

    set bytewidth_var [expr [get_parameter_value AXI_DATA_W]/8]
    set_parameter_value AXI_NUMBYTES $bytewidth_var
}
```

#### Related Links

- [Declare Parameters with Custom `\_hw.tcl` Commands](#) on page 630
- [Validate Parameter Values with a Validation Callback](#) on page 632
- [Component Interface Tcl Reference](#) on page 791

## 10.13 Control File Generation Dynamically with Parameters and a Fileset Callback

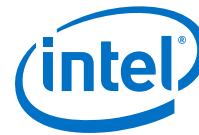
You can use a fileset callback to control which files are created in the output directories during the generation phase based on parameter values, instead of providing a fixed list of files. In a callback procedure, you can query the values of the parameters and use them to generate the appropriate files. To define a fileset callback, you specify a callback procedure name as an argument in the `add_fileset` command. You can use the same fileset callback procedure for all of the filesets, or create separate procedures for synthesis and simulation, or Verilog and VHDL.

### Example 92. Fileset Callback Using Parameters to Control Filesets in Two Different Ways

The `RAM_VERSION` parameter chooses between two different source files to control the implementation of a RAM block. For the top-level source file, a custom Tcl routine generates HDL that optionally includes control and status registers, depending on the value of the `CSR_ENABLED` parameter.

During the generation phase, Platform Designer creates a a top-level Platform Designer system HDL wrapper module to instantiate the component top-level module, and applies the component's parameters, for any parameter whose parameter property `HDL_PARAMETER` is set to true.

```
#Create synthesis fileset with fileset_callback and set top level
add_fileset my_synthesis_fileset QUARTUS_SYNTH fileset_callback
```



```

set_fileset_property my_synthesis_fileset TOP_LEVEL \
demo_axi_memory

# Create Verilog simulation fileset with same fileset_callback
# and set top level

add_fileset my_verilog_sim_fileset SIM_VERILOG fileset_callback

set_fileset_property my_verilog_sim_fileset TOP_LEVEL \
demo_axi_memory

# Add extra file needed for simulation only

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv

# Create VHDL simulation fileset (with Verilog files
# for mixed-language VHDL simulation)

add_fileset my_vhdl_sim_fileset SIM_VHDL fileset_callback
set_fileset_property my_vhdl_sim_fileset TOP_LEVEL demo_axi_memory

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH
verification_lib/verbosity_pkg.sv

# Define parameters required for fileset_callback

add_parameter RAM_VERSION INTEGER 1
set_parameter_property RAM_VERSION ALLOWED_RANGES {1 2}
set_parameter_property RAM_VERSION HDL_PARAMETER false
add_parameter CSR_ENABLED BOOLEAN enable
set_parameter_property CSR_ENABLED HDL_PARAMETER false

# Create Tcl callback procedure to add appropriate files to
# filesets based on parameters

proc fileset_callback { entityName } {
    send_message INFO "Generating top-level entity $entityName"
    set ram [get_parameter_value RAM_VERSION]
    set csr_enabled [get_parameter_value CSR_ENABLED]

    send_message INFO "Generating memory
implementation based on RAM_VERSION $ram      "

        if {$ram == 1} {
            add_fileset_file single_clk_ram1.v VERILOG PATH \
single_clk_ram1.v
        } else {
            add_fileset_file single_clk_ram2.v VERILOG PATH \
single_clk_ram2.v
        }

    send_message INFO "Generating top-level file for \
CSR_ENABLED $csr_enabled"

    generate_my_custom_hdl $csr_enabled demo_axi_memory_gen.sv

    add_fileset_file demo_axi_memory_gen.sv VERILOG PATH \
demo_axi_memory_gen.sv
}

```

### Related Links

- [Specify Synthesis and Simulation Files in the Platform Designer Component Editor on page 618](#)
- [Component Interface Tcl Reference on page 791](#)

## 10.14 Create a Composed Component or Subsystem

A composed component is a subsystem containing instances of other components. Unlike an HDL-based component, a composed component's HDL is created by generating HDL for the components in the subsystem, in addition to the Platform Designer interconnect to connect the subsystem instances.

You can add child instances in a composition callback of the `_hw.tcl` file.

With a composition callback, you can also instantiate and parameterize sub-components as a function of the composed component's parameter values. You define a composition callback by setting the `COMPOSITION_CALLBACK` module property to the name of the composition callback procedures.

A composition callback replaces the validation and elaboration phases. HDL for the subsystem is generated by generating all of the sub-components and the top-level that combines them.

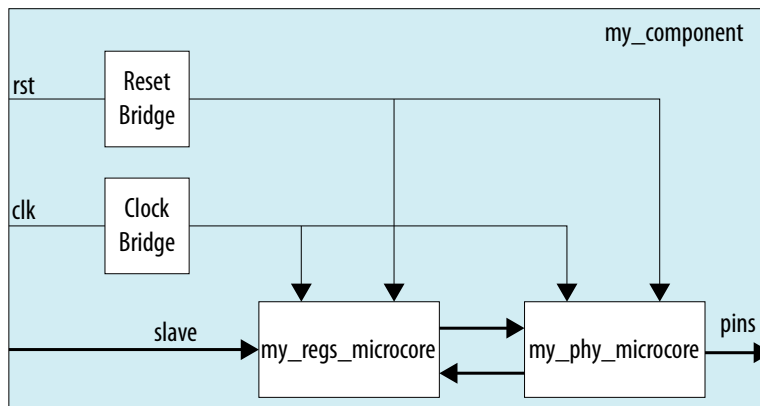
To connect instances of your component, you must define the component's interfaces. Unlike an HDL-based component, a composed component does not directly specify the signals that are exported. Instead, interfaces of submodules are chosen as the external interface, and each internal interface's ports are connected through the exported interface.

Exporting an interface means that you are making the interface visible from the outside of your component, instead of connecting it internally. You can set the `EXPORT_OF` property of the externally visible interface from the main program or the composition callback, to indicate that it is an exported view of the submodule's interface.

Exporting an interface is different than defining an interface. An exported interface is an exact copy of the subcomponent's interface, and you are not allowed to change properties on the exported interface. For example, if the internal interface is a 32-bit or 64-bit master without bursting, then the exported interface is the same. An interface on a subcomponent cannot be exported and also connected within the subsystem.

When you create an exported interface, the properties of the exported interface are copied from the subcomponent's interface without modification. Ports are copied from the subcomponent's interface with only one modification; the names of the exported ports on the composed component are chosen to ensure that they are unique.

Figure 195. Top-Level of a Composed Component



Example 93. Composed `_hw.tcl` File that Instantiates Two Sub-Components

Platform Designer connects the components, and also connects the clocks and resets. Note that clock and reset bridge components are required to allow both sub-components to see common clock and reset inputs.

```
package require -exact qsys 14.0
set_module_property name my_component
set_module_property COMPOSITION_CALLBACK composed_component

proc composed_component {} {
  add_instance clk altera_clock_bridge
  add_instance reset altera_reset_bridge
  add_instance regs my_regs_microcore
  add_instance phy my_phy_microcore

  add_interface clk clock end
  add_interface reset reset end
  add_interface slave avalon slave
  add_interface pins conduit end

  set_interface_property clk EXPORT_OF clk.in_clk
  set_instance_property_value reset synchronous_edges deassert
  set_interface_property reset EXPORT_OF reset.in_reset
  set_interface_property slave EXPORT_OF regs.slave
  set_interface_property pins EXPORT_OF phy.pins

  add_connection clk.out_clk reset.clk
  add_connection clk.out_clk regs.clk
  add_connection clk.out_clk phy.clk
  add_connection reset.out_reset regs.reset
  add_connection reset.out_reset phy.clk_reset
  add_connection regs.output phy.input
  add_connection phy.output regs.input
}
```

### Related Links

[Component Interface Tcl Reference](#) on page 791

## 10.15 Add Component Instances to a Static or Generated Component

You can create nested components by adding component instances to an existing component. Both static and generated components can create instances of other components. You can add child instances of a component in a `_hw.tcl` using elaboration callback.

With an elaboration callback, you can also instantiate and parameterize sub-components with the `add_hdl_instance` command as a function of the parent component's parameter values.

When you instantiate multiple nested components, you must create a unique variation name for each component with the `add_hdl_instance` command. Prefixing a variation name with the parent component name prevents conflicts in a system. The variation name can be the same across multiple parent components if the generated parameterization of the nested component is exactly the same.

**Note:** If you do not adhere to the above naming variation guidelines, Platform Designer validation-time errors occur, which are often difficult to debug.

### Related Links

- [Static Components](#) on page 640
- [Generated Components](#) on page 641

### 10.15.1 Static Components

Static components always generate the same output, regardless of their parameterization. Components that instantiate static components must have only static children.

A design file that is static between all parameterizations of a component can only instantiate other static design files. Since static IPs always render the same HDL regardless of parameterization, Platform Designer generates static IPs only once across multiple instantiations, meaning they have the same top-level name set.

#### Example 94. Typical Usage of the `add_hdl_instance` Command for Static Components

```
package require -exact qsys 14.0

set_module_property name add_hdl_instance_example
add_fileset synth_fileset QUARTUS_SYNTH synth_callback
set_fileset_property synth_fileset TOP_LEVEL basic_static
set_module_property elaboration_callback elab

proc elab {} {
    # Actual API to instantiate an IP Core
    add_hdl_instance emif_instance_name altera_mem_if_ddr3_emif

    # Make sure the parameters are set appropriately
    set_instance_parameter_value emif_instance_name SPEED_GRADE {7}
    ...
}

proc synth_callback { output_name } {
    add_fileset_file "basic_static.v" VERILOG PATH basic_static.v
}
```





### Example 95. Top-Level HDL Instance and Wrapper File Created by Platform Designer

In this example, Platform Designer generates a wrapper file for the instance name specified in the `_hw.tcl` file.

```
//Top Level Component HDL
module basic_static (input_wire, output_wire, inout_wire);
input [31:0] input_wire;
output [31:0] output_wire;
inout [31:0] inout_wire;

// Instantiation of the instance added via add_hdl_instance
// command. This is an example of how the instance added via
// the add_hdl_instance command can be used
// in the top-level file of the component.

emif_instance_name fixed_name_instantiation_in_top_level(
.pll_ref_clk (input_wire), // pll_ref_clk.clk
.global_reset_n (input_wire), // global_reset.reset_n
.soft_reset_n (input_wire), // soft_reset.reset_n
...
... );
endmodule

//Wrapper for added HDL instance
// emif_instance_name.v
// Generated using ACDS version 14.0

`timescale 1 ps / 1 ps
module emif_instance_name (
input wire pll_ref_clk, // pll_ref_clk.clk
input wire global_reset_n, // global_reset.reset_n
input wire soft_reset_n, // soft_reset.reset_n
output wire afi_clk, // afi_clk.clk
...
...);
example_addhdlinstance_system
_add_hdl_instance_example_0_emif_instance
_name_emif_instance_name emif_instance_name (

.pll_ref_clk (pll_ref_clk), // pll_ref_clk.clk
.global_reset_n (global_reset_n), // global_reset.reset_n
.soft_reset_n (soft_reset_n), // soft_reset.reset_n
...
...);
endmodule
```

### 10.15.2 Generated Components

A generated component's fileset callback allows an instance of the component to create unique HDL design files based on the instance's parameter values. For example, you can write a fileset callback to include a control and status interface based on the value of a parameter. The callback overcomes a limitation of HDL languages, which do not allow run-time parameters.

Generated components change their generation output (HDL) based on their parameterization. If a component is generated, then any component that may instantiate it with multiple parameter sets must also be considered generated, since its HDL changes with its parameterization. This case has an effect that propagates up to the top-level of a design.

Since generated components are generated for each unique parameterized instantiation, when implementing the `add_hdl_instance` command, you cannot use the same fixed name (specified using `instance_name`) for the different variants of

the child HDL instances. To facilitate unique naming for the wrapper of each unique parameterized instantiation of child HDL instances, you must use the following command so that Platform Designer generates a unique name for each wrapper. You can then access this unique wrapper name with a fileset callback so that the instances are instantiated inside the component's top-level HDL.

- To declare auto-generated fixed names for wrappers, use the command:

```
set_instance_property instance_name HDLINSTANCE_USE_GENERATED_NAME \
true
```

*Note:* You can only use this command with a generated component in the global context, or in an elaboration callback.

- To obtain auto-generated fixed name with a fileset callback, use the command:

```
get_instance_property instance_name HDLINSTANCE_GET_GENERATED_NAME
```

*Note:* You can only use this command with a fileset callback. This command returns the value of the auto-generated fixed name, which you can then use to instantiate inside the top-level HDL.

### Example 96. Typical Usage of the `add_hdl_instance` Command for Generated Components

Platform Designer generates a wrapper file for the instance name specified in the `_hw.tcl` file.

```
package require -exact qsys 14.0
set_module_property name generated_toplevel_component
set_module_property ELABORATION_CALLBACK elaborate
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH generate
add_fileset SIM_VERILOG SIM_VERILOG generate
add_fileset SIM_VHDL SIM_VHDL generate

proc elaborate {} {
    # Actual API to instantiate an IP Core
    add_hdl_instance emif_instance_name altera_mem_if_ddr3_emif

    # Make sure the parameters are set appropriately
    set_instance_parameter_value emif_instance_name SPEED_GRADE {7}
    ...
    # instruct Platform Designer to use auto generated fixed name
    set_instance_property emif_instance_name \
        HDLINSTANCE_USE_GENERATED_NAME 1
}

proc generate { entity_name } {
    # get the autogenerated name for emif_instance_name added
    # via add_hdl_instance

    set autogeneratedfixedname [get_instance_property \
        emif_instance_name HDLINSTANCE_GET_GENERATED_NAME]

    set fileID [open "generated_toplevel_component.v" r]
    set temp ""

    # read the contents of the file

    while {[eof $fileID] != 1} {
        gets $fileID lineInfo

        # replace the top level entity name with the name provided
        # during generation

        regsub -all "substitute_entity_name_here" $lineInfo \
```



```
"${entity_name}" lineInfo

# replace the autogenerated name for emif_instance_name added
# via add_hdl_instance

regsub -all "substitute_autogenerated_emifinstancename_here" \
$lineInfo"${autogeneratedfixedname}" lineInfo \
append temp "${lineInfo}\n"
}

# adding a top level component file

add_fileset_file ${entity_name}.v VERILOG TEXT $temp
}
```

### Example 97. Top-Level HDL Instance and Wrapper File Created By Platform Designer

```
// Top Level Component HDL

module substitute_entity_name_here (input_wire, output_wire,
inout_wire);

input [31:0] input_wire;
output [31:0] output_wire;
inout [31:0] inout_wire;

// Instantiation of the instance added via add_hdl_instance
// command. This is an example of how the instance added
// via add_hdl_instance command can be used
// in the top-level file of the component.

substitute_autogenerated_emifinstancename_here
fixed_name_instantiation_in_top_level (
.pll_ref_clk (input_wire), // pll_ref_clk.clk
.global_reset_n (input_wire), // global_reset.reset_n
.soft_reset_n (input_wire), // soft_reset.reset_n
...
... );
endmodule

// Wrapper for added HDL instance
// generated_toplevel_component_0_emif_instance_name.v is the
// auto generated //emif_instance_name
// Generated using ACDS version 13.

`timescale 1 ps / 1 ps
module generated_toplevel_component_0_emif_instance_name (
input wire pll_ref_clk, // pll_ref_clk.clk
input wire global_reset_n, // global_reset.reset_n
input wire soft_reset_n, // soft_reset.reset_n
output wire afi_clk, // afi_clk.clk
...
...);
example_addhdlinstance_system_add_hdl_instance_example_0_emif
_instance_name_emif_instance_name emif_instance_name (

.pll_ref_clk (pll_ref_clk), // pll_ref_clk.clk
.global_reset_n (global_reset_n), // global_reset.reset_n
.soft_reset_n (soft_reset_n), // soft_reset.reset_n
...
...);
endmodule
```

### Related Links

- [Control File Generation Dynamically with Parameters and a Fileset Callback](#) on page 636
- [Intellectual Property & Reference Designs](#)

### 10.15.3 Design Guidelines for Adding Component Instances

In order to promote standard and predictable results when generating static and generated components, Intel recommends the following best-practices:

- For two different parameterizations of a component, a component must never generate a file of the same name with different instantiations. The contents of a file of the same name must be identical for every parameterization of the component.
- If a component generates a nested component, it must never instantiate two different parameterizations of the nested component using the same instance name. If the parent component's parameterization affects the parameters of the nested component, the parent component must use a unique instance name for each unique parameterization of the nested component.
- Static components that generate differently based on parameterization have the potential to cause problems in the following cases:
  - Different file names with the same entity names, results in same entity conflicts at compilation-time
  - Different contents with the same file name results in overwriting other instances of the component, and in either file, compile-time conflicts or unexpected behavior.
- Generated components that generate files not based on the output name and that have different content results in either compile-time conflicts, or unexpected behavior.

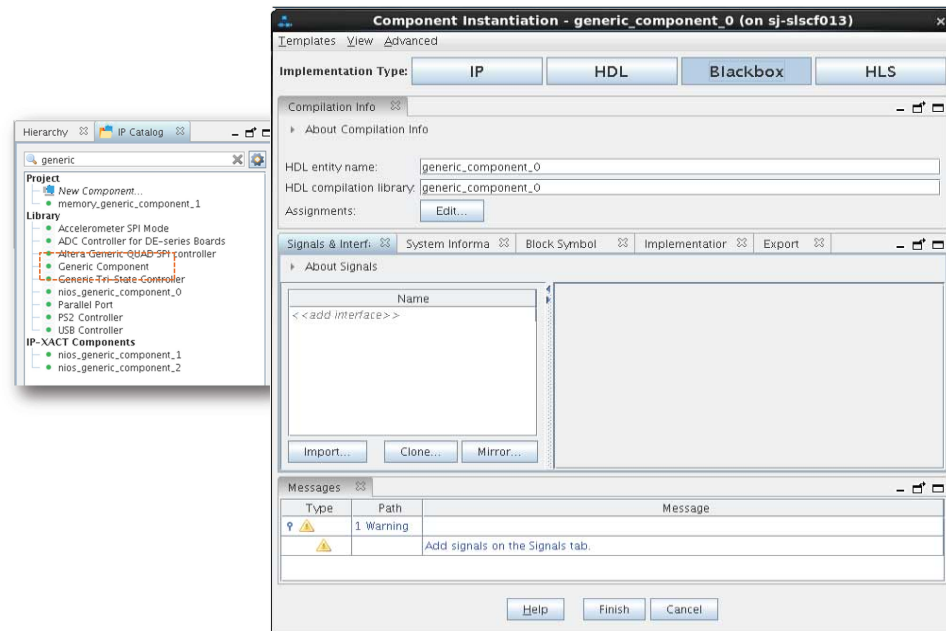
## 10.16 Adding a Generic Component to the Platform Designer System

The generic component is a type of Platform Designer component that enables hierarchical isolation of IP components. This component is available in the IP Catalog. Use this component as a mechanism to quickly define a custom component or import your RTL into a Platform Designer system.

By default, the generic component's **Implementation Type** is set to **Blackbox**. This mode specifies that the RTL implementation is not provided in the generated RTL output of the Platform Designer system. When you generate a system containing a generic component, the system's RTL instantiates the component, but does not provide an implementation for the component. You must provide an implementation for the component in a downstream compiler such as Intel Quartus Prime software or RTL code.



Figure 196. Adding a Generic Component to the Platform Designer System



To add a generic component to your system:

1. Type `generic` component in the IP Catalog.
2. To launch the **Component Instantiation** editor, double-click **Generic Component**. The default option is to create a **Blackbox** component.

The **Component Instantiation** editor allows you to select one of four implementation types:

- **IP**—Use the **IP** option to create a component from a `.ip` file.
- **HDL**—Use the **HDL** option to instantiate a component from RTL (`.v`/`.sv`/`.vhd`) without using `_hw.tcl`.
- **Blackbox**—The default option. Use the **Blackbox** option to create a generic component. You can either add interfaces and signals manually, clone/mirror from existing components in the current system, or import from an `.ipxact` file.
- **HLS**—Use the **HLS** option to add and compile High Level Synthesis (HLS) files, or add and import HLS files.

### Related Links

- [Creating Custom Interfaces in a Generic Component](#) on page 646
- [Instantiating RTL in a System as a Generic Component](#) on page 649
- [Implementing Generic Components Using High Level Synthesis Files](#) on page 650

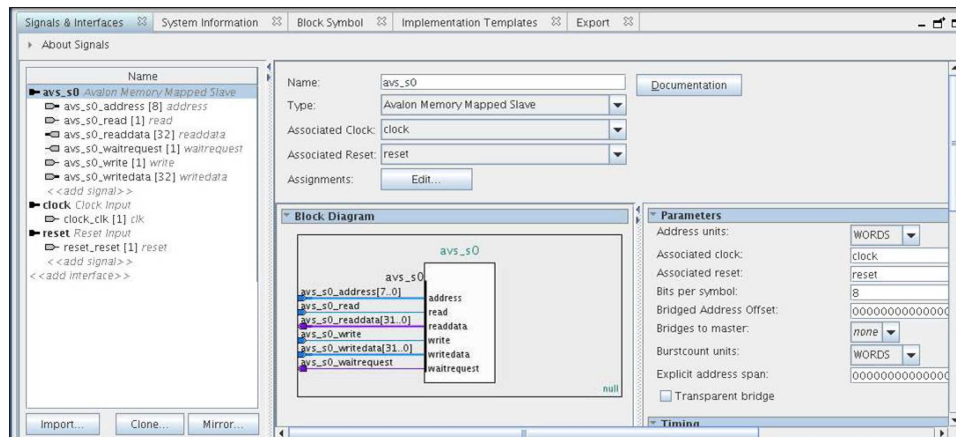
### 10.16.1 Creating Custom Interfaces in a Generic Component

The **Signals & Interfaces** tab of the **Component Instantiation** editor allows you to customize signals and interfaces for your generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. To add an interface, click <<add interface>> in the left pane and select the interface. The selected interface appears in the parameter editor to the right, where you specify its parameters.
4. To add signals to the selected interface, click <<add signal>> below the selected interface.
5. To move signals between interfaces, select the signal and drag it to another interface.
6. To rename a signal or interface, select the element, and then press F2.
7. To remove a signal or interface, right-click the element, and then click **Remove**.

*Note:* Alternatively, to remove a signal or interface, select the element and press **Delete**. When you remove an interface, Platform Designer also removes all of its associated signals.

**Figure 197. Creating Custom Interfaces**



*Note:* To add existing template interfaces to your generic component, select the interface from **Templates** menu in the **Component Instantiation** editor.

#### 10.16.1.1 Mirroring Interfaces in a Generic Component

To mirror existing signals and interfaces from an IP component to your generic component:



1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Mirror** button. A list appears which lists all the available components in the system and their associated interfaces.
4. Select the desired interface. Platform Designer mirrors the interface and its associated signals and adds the mirrored interfaces and signals to the **Signals & Interfaces** tab of the generic component.

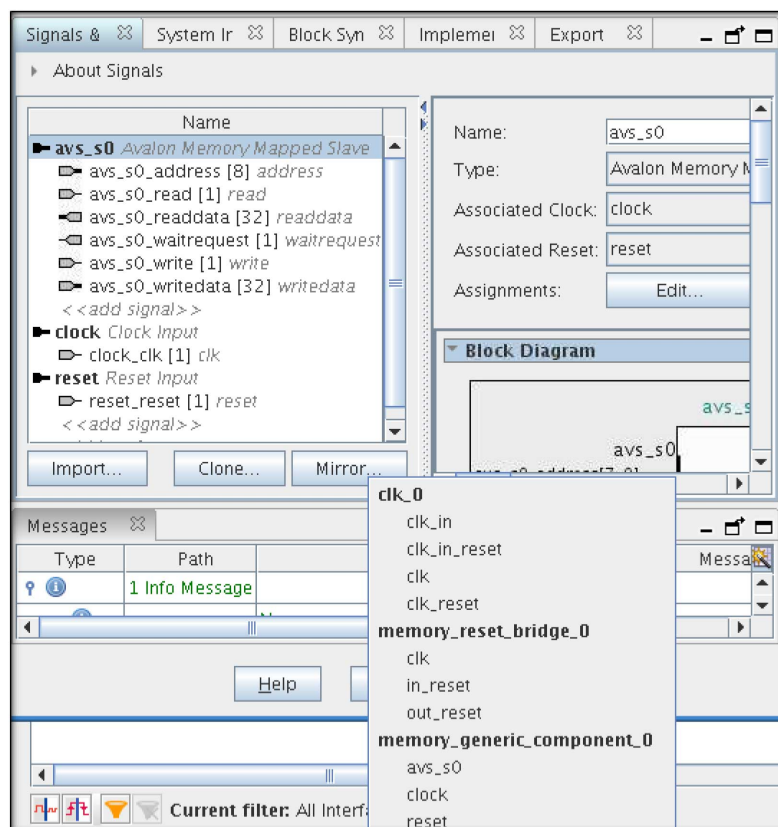
**Example 98. Mirroring Interfaces in a Generic Component Example**

Selected Interface	Mirrored Interface
Avalon Memory-Mapped Master (avalon_master)	Avalon Memory-Mapped Slave (avalon_slave)

Signals of the Selected Interface	Signals of the Mirrored Interface
waitrequest(Input 1)	waitrequest(Output 1)
readdata(Input 32)	readdata(Output 32)
readdatavalid(Input 1)	readdatavalid(Output 1)
burstcount(Output 32)	burstcount(Input 32)

**Figure 198. Mirroring Interfaces**

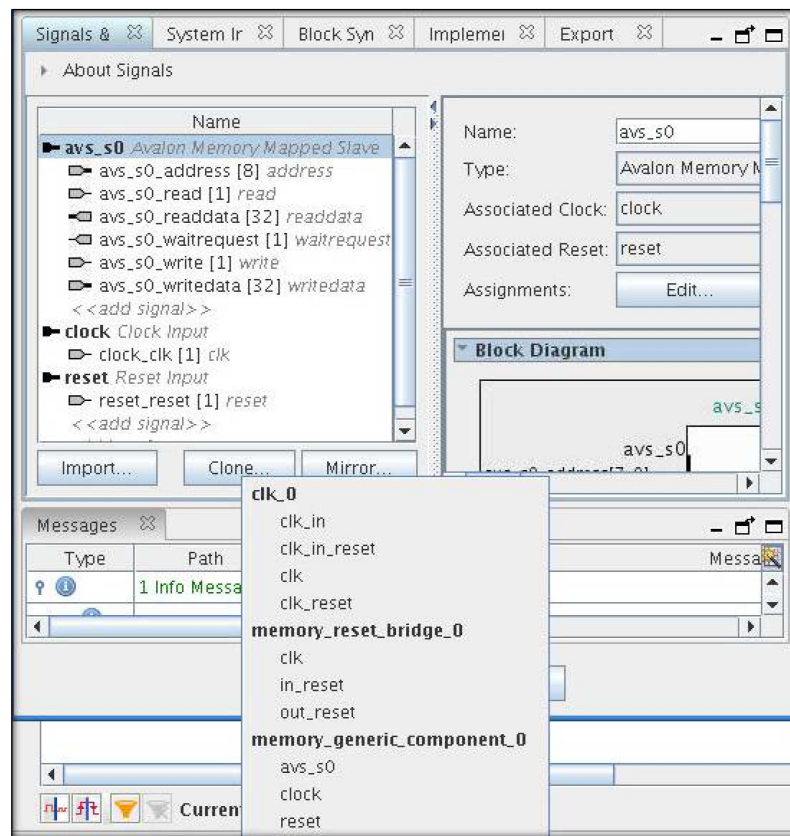


### 10.16.1.2 Cloning Interfaces in a Generic Component

To clone existing signals and interfaces from an IP component to your generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Clone** button. A list appears which lists all the available components in the system and their associated interfaces.
4. Select the desired interface. Platform Designer clones the interface and adds an exact replica of the interface and its associated signals to the **Signals & Interfaces** tab of the generic component.

Figure 199. Cloning Interfaces



### 10.16.1.3 Importing Interfaces to a Generic Component

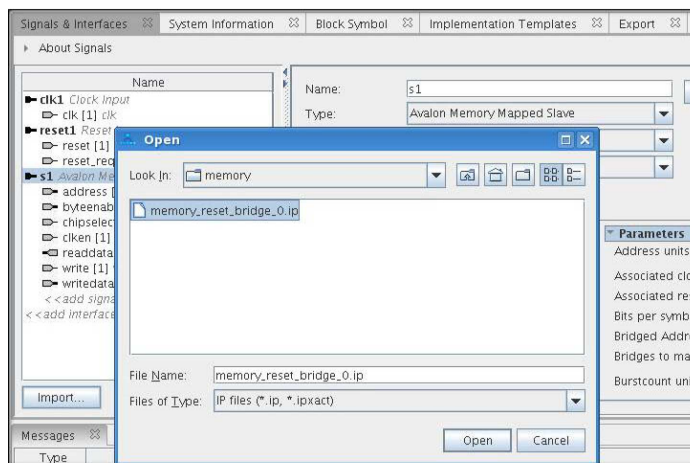
To import interfaces from an existing IP or IP-XACT file to your generic component:





1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Import** button. A dialog box appears from where you choose the IP/IP-XACT file to import to your generic component.
4. Select the desired interface. Platform Designer populates the **Signals & Interfaces** tab with the signals and interfaces defined in the selected file.

Figure 200. Importing Interfaces



### 10.16.2 Instantiating RTL in a System as a Generic Component

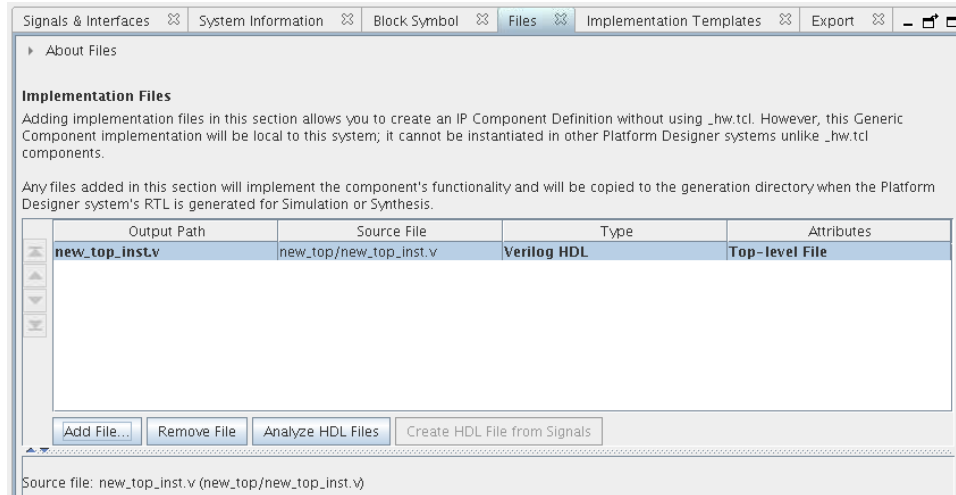
To add an RTL file as a generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, set the **Implementation Type** as **HDL**.
3. Select the **Files** tab.
4. Click **Add File** and select the RTL file to load to the generic component.
  - a. If you are importing an HDL file with SystemVerilog interface definition, you should set the **Attributes** of this file to **File contains SystemVerilog interface definition used by the Top-level Module**.
5. Click **Analyze HDL files**. This option analyzes and populates the **Signals & Interfaces** tab of the generic component from the RTL file.
6. Verify, and modify the signals and interfaces if needed, in the **Signals & Interfaces** tab.

*Note:*

You must treat a generic component with an **HDL Implementation Type** as a customized and centralized RTL, specific to your current system. When you set a generic component's **Implementation Type** to **HDL**, the output of any RTL that you add to the component is within the system's output directory.

Figure 201. Instantiating an RTL as a Generic Component



### 10.16.3 Implementing Generic Components Using High Level Synthesis Files

High Level Synthesis (HLS) files can be compiled to create Platform Designer components and are written according to the i++ specification. HLS files can be in \*.c, \*.cc, \*.cpp, \*.c++, \*.cp, or \*.cxx format.

An HLS file defines one or more components in an i++ format that Platform Designer compiles into HDL. In order to add components from an HLS file there are two basic steps:

1. Identify and add the HLS file.
2. **Import** an already compiled file from a previous Platform Designer session or project, or **Compile** the HLS file in Platform Designer.

Once the component has been imported or compiled, Platform Designer performs the following actions:

- Imports an `.ip` resulting from the HLS compilation to the component name defined in the HLS file.
- Sets the **HDL entity name** and **HDL compilation library** to the component defined in the HLS file.
- Adds the `.ip` file to the empty generic component.
- Adds paths to the `.ip` and `_hw.tcl` output files to the Platform Designer search path to enable instantiation.
- Populates the signals and interfaces of the component from the `.ip` file.



After compilation, the HLS compiler creates a `<component_name>.prj` folder with the following directories:

**Table 125. Contents of `<component_name>.prj` Folder**

Folder	Description
<code>/component</code>	Contains IP and <code>_hw.tcl</code> files.
<code>/quartus</code>	Contains Intel Quartus Prime Pro Edition project files that instantiate the HLS component. You can use this to verify timing and logic usage.
<code>/reports</code>	Contains a compilation report in HTML.
<code>/verification</code>	Contains verification files, if you decided to create a verification executable.

### Related Links

[Intel High Level Synthesis Compiler Getting Started Guide](#)

#### 10.16.3.1 Add High Level Synthesis Files to a Generic Component

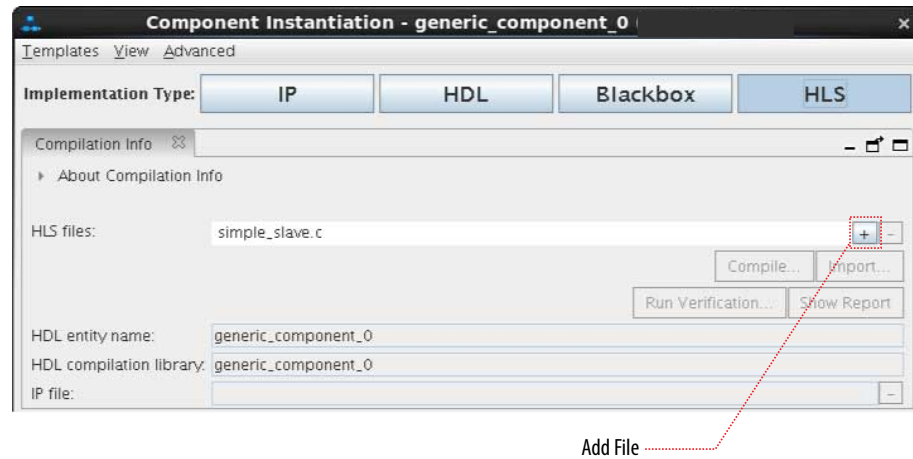
You can quickly add High Level Synthesis (HLS) components to a Platform Designer project by dragging and dropping files into the Platform Designer **System Contents** tab. The drag-and-drop process selects the **HLS** implementation type, and adds the HLS file to the **HLS files** box.

To add a component with an HLS implementation, perform the following steps in Platform Designer:

1. Drag an HLS file to the **System Contents** tab of Platform Designer.  
or
2. Type `generic component` in the IP Catalog.
3. To launch the **Component Instantiation** editor, double-click **Generic Component**.
4. To add a component from an HLS file to the empty generic component, select the **HLS Implementation Type**.
5. Click **+** and select an HLS file to add.

You can click **+** to add more than one HLS file. Click **-** to remove HLS files. The primary case for adding multiple HLS files is when you are using a library of components defined by one or more high level synthesis files.

Figure 202. Add an HLS Implementation Type File



### Related Links

- [Compile High Level Synthesis Files](#) on page 652
- [Import High Level Synthesis Files](#) on page 654

### 10.16.3.2 Compile High Level Synthesis Files

The **Compile** option for High Level Synthesis (HLS) component instantiation in Platform Designer invokes the Intel HLS Compiler to compile HLS files and modify a generic component.

Performing a compile on an HLS file has the following results:

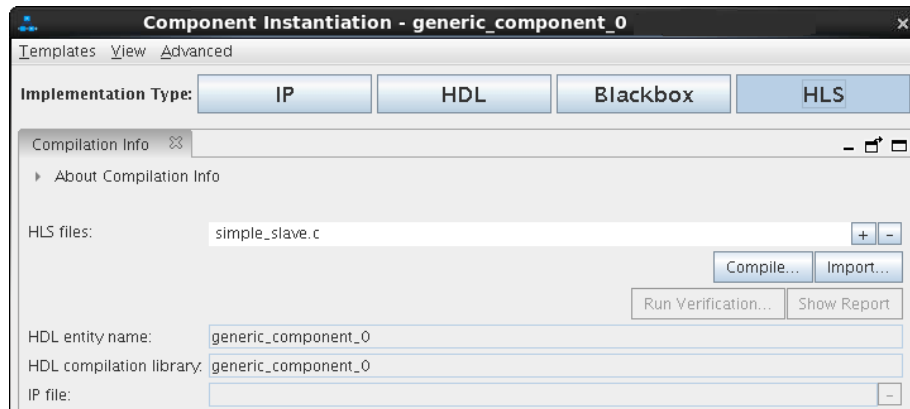
- Imports an `.ip` resulting from the HLS compilation to the component name defined in the HLS file.
- Sets the **HDL entity name** and **HDL compilation library** to the component defined in the HLS file.
- Adds the `.ip` file to the empty generic component.
- Adds paths to the `.ip` and `_hw.tcl` output files to the Platform Designer search path to enable instantiation.
- Populates the signals and interfaces of the component from the `.ip` file.

After you have added an HLS file:

1. Click **Compile**.

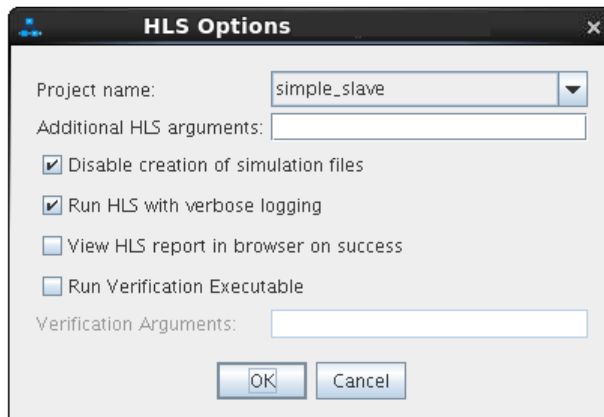


Figure 203. HLS Component Instantiation



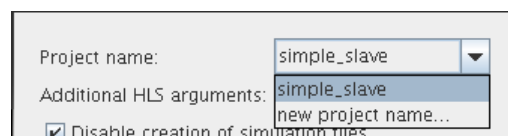
2. In the **HLS Options** dialog box, you can select from the following options:

Figure 204. HLS Options Dialog Box



- a. The project name defaults to the entity name defined in the HLS file. To set a new project name, select **new project name** and enter a new HLS project name in the dialog box.

Figure 205. Change the Project Name



- b. Provide additional arguments to the HLS compiler. Refer to *Command Compiler Options* in the *Intel High Level Synthesis Reference Manual* for information on compiler arguments.
- c. Disable or enable simulation file creation. A simulation file is required to use the **Run Verification** option after compilation is complete.
- d. Enable verbose logging to create a compilation log file.

- e. Enable or disable display of the HLS report in a browser window directly after compilation is complete.
  - f. Perform verification with or without additional verification arguments if you chose to create a verification executable. Refer to the *Intel High Level Synthesis Compiler User Guide* for information on verification arguments.
3. Click **OK** to compile the HLS file and create the component.
  4. If your HLS file defines more than one component, the **Choose File to Import** dialog box prompts you to select a specific component from a list.
  5. After compiling, click **Show Report** to display a compilation report in a browser window.
  6. If you created simulation files for your component, you can click **Run Verification** to perform verification.

**Related Links**

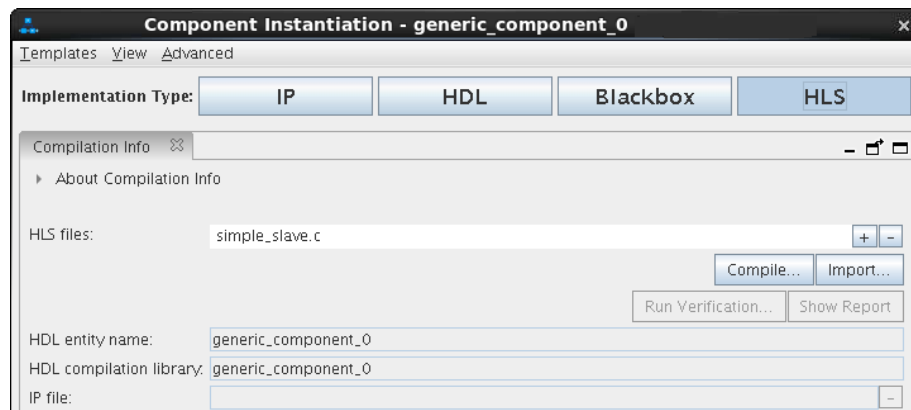
- [Compiler Command Options](#)
- [Intel High Level Synthesis Compiler User Guide](#)

**10.16.3.3 Import High Level Synthesis Files**

If you have a compiled High Level Synthesis (HLS) file, you can import it instead to save compilation time.

1. Click **Import**.

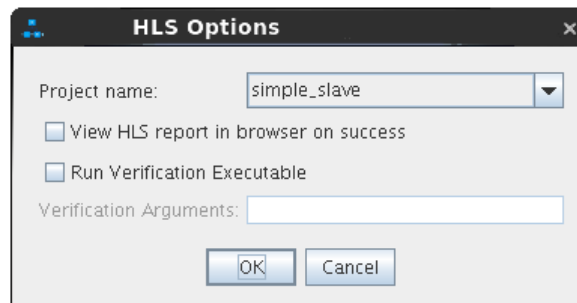
**Figure 206. HLS Component Instantiation**



You should only use **Import** when your HLS file defines previously compiled components.

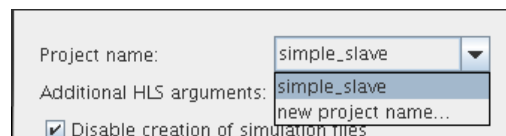


Figure 207. HLS Options Dialog Box



2. In the **HLS Options** dialog box, you can select from the following options:
  - a. The project name defaults to the entity name defined in the HLS file. To set a new project name, select **new project name** and enter a new HLS project name in the dialog box.

Figure 208. Change the Project Name



- b. Enable or disable display of the HLS report in a browser window directly after compilation is complete.
    - c. Perform verification with or without additional verification arguments if you chose to create a verification executable. Refer to the *Intel High Level Synthesis Compiler User Guide* for information on verification arguments.
3. Click **OK**.
4. If your HLS file defines more than one component, the **Choose File to Import** dialog box prompts you to select a specific component `.ip` from a list.
5. After importing, click **Show Report** to display a compilation report in a browser window if the compilation report is enabled.
6. Click **Run Verification** to perform verification if it is enabled.

#### Related Links

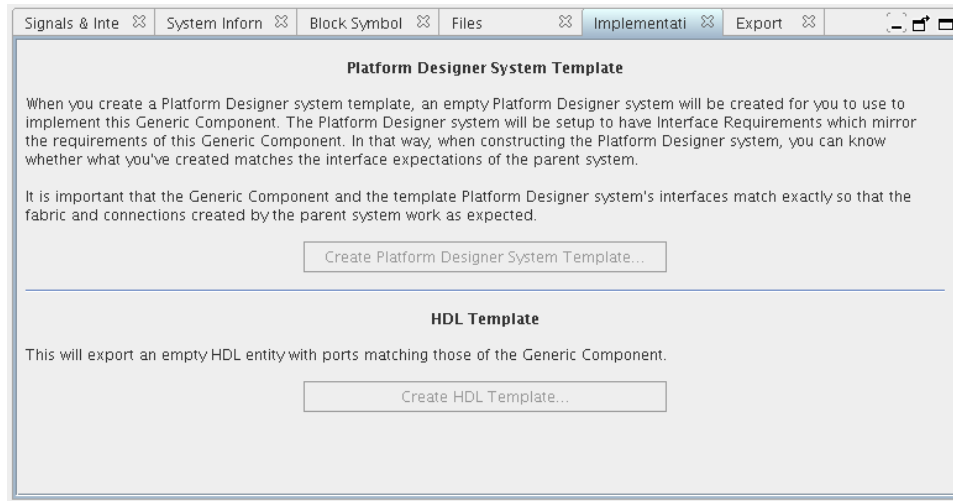
[Intel High Level Synthesis Compiler User Guide](#)

### 10.16.4 Creating System Template for a Generic Component

To create a Platform Designer system template:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, add the interfaces and signals for the new component in the **Signals & Interfaces** tab.
3. Select the **Implementation Templates** tab.
4. Click **Create Platform Designer System Template** button. This option creates an empty Platform Designer system and saves the template as a `.qsys` file to implement this generic component.

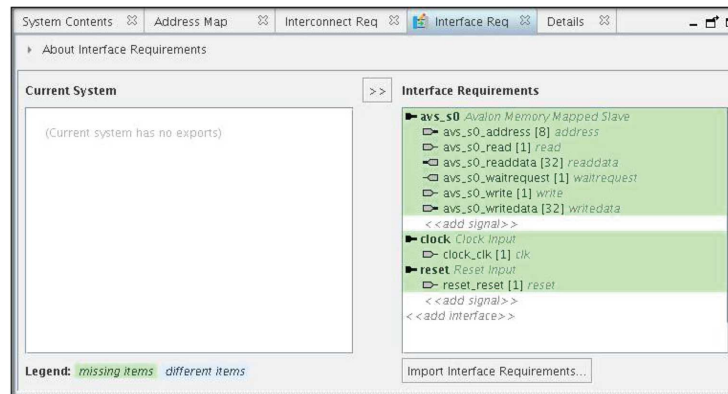
Figure 209. Creating System templates



To implement this component:

1. To open the template Platform Designer system, click **File** ► **Open** and choose the specific .qsys file.
2. Add either or both IP components and generic components then export their interfaces to satisfy the specified interface requirements.
3. To view the exported interfaces in the **Interface Requirements** tab, select **View** ► **Interface Requirements**.

Figure 210. Viewing the Interface Requirements from the System Template







### 10.16.5 Exporting a Generic Component

You can export a generic component as a `.ipxact` file as well as `_hw.tcl` file:

1. Double-click **Generic Component** in the IP Catalog.
2. Select the **Export** tab.
3. To export generic component as an IP-XACT file, click **Export IP-XACT File** and select the location to save your IP-XACT file.
4. To export generic component as a `_hw.tcl` file, click **Export \_hw.tcl File** and select the location to save your `_hw.tcl` file.

## 10.17 Document Revision History

The table below indicates edits made to the *Creating Platform Designer Components* content since its creation.

**Table 126. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>• Changed instances of <i>Qsys Pro</i> to Platform Designer</li> <li>• Replaced mentions of <code>altera_axi_default_slave</code> to <code>altera_error_response_slave</code></li> <li>• Added support for SystemVerilog interfaces with <code>_hw.tcl</code>.</li> <li>• Added support for user alterable HDL parameters with <code>_hw.tcl</code>.</li> <li>• Added support for High Level Synthesis file compilation.</li> </ul>
2017.05.08	17.0.0	<ul style="list-style-type: none"> <li>• Updated Figure: Address Span Extender</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>• Implemented Intel rebranding.</li> <li>• Implemented Platform Designer rebranding.</li> <li>• Added topics for Generic Component.</li> </ul>
2015.11.02	15.1.0	Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i> .
2015.05.04	15.0.0	<ul style="list-style-type: none"> <li>• Updated screen shots <b>Files</b> tab, Platform Designer Component Editor.</li> <li>• Added topic: <i>Specify Interfaces and Signals in the Platform Designer Component Editor</i>.</li> <li>• Added topic: <i>Create an HDL File in the Platform Designer Component Editor</i>.</li> <li>• Added topic: <i>Create an HDL File Using a Template in the Platform Designer Component Editor</i>.</li> </ul>
November 2013	13.1.0	<ul style="list-style-type: none"> <li>• <code>add_hdl_instance</code></li> <li>• Added <i>Creating a Component With Differing Structural Platform Designer View and Generated Output Files</i>.</li> </ul>
May 2013	13.0.0	<ul style="list-style-type: none"> <li>• Consolidated content from other Platform Designer chapters.</li> <li>• Added <i>Upgrading IP Components to the Latest Version</i>.</li> <li>• Updated for AMBA APB support.</li> </ul>
November 2012	12.1.0	<ul style="list-style-type: none"> <li>• Added AMBA AXI4 support.</li> <li>• Added the <b>demo_axi_memory</b> example with screen shots and example <code>_hw.tcl</code> code.</li> </ul>
June 2012	12.0.0	<ul style="list-style-type: none"> <li>• Added new tab structure for the Component Editor.</li> <li>• Added AMBA 3 AXI support.</li> </ul>
November 2011	11.1.0	Template update.
May 2011	11.0.0	<ul style="list-style-type: none"> <li>• Removed beta status.</li> <li>• Added Avalon Tri-state Conduit (Avalon-TC) interface type.</li> <li>• Added many interface templates for Nios custom instructions and Avalon-TC interfaces.</li> </ul>
December 2010	10.1.0	Initial release.



### **Related Links**

#### [Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 11 Platform Designer Interconnect

---

Platform Designer interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.

*Note:* Intel now refers to Qsys Pro as Platform Designer.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

*Note:* The video *AMBA AXI and Intel Avalon Interoperation Using Platform Designer*, describes seamless integration of IP components using the AMBA AXI interface, and the Intel Avalon interface.

### Related Links

- [Creating a System with Platform Designer](#) on page 327
- [Creating Platform Designer Components](#) on page 608
- [Platform Designer System Design Components](#) on page 914
- [AMBA AXI and Intel Avalon Interoperation Using Platform Designer](#)
- [Avalon Interface Specifications](#)

### 11.1 Memory-Mapped Interfaces

Platform Designer supports the implementation of memory-mapped interfaces for Avalon, AXI, and APB protocols.

Platform Designer interconnect transmits memory-mapped transactions between masters and slaves in packets. The command network transports read and write packets from master interfaces to slave interfaces. The response network transports response packets from slave interfaces to master interfaces.

For each component interface, Platform Designer interconnect manages memory-mapped transfers and interacts with signals on the connected interface. Master and slave interfaces can implement different signals based on interface parameterizations, and Platform Designer interconnect provides any necessary adaptation between them. In the path between master and slaves, Platform Designer interconnect may introduce registers for timing synchronization, finite state machines for event sequencing, or nothing at all, depending on the services required by the interfaces.



Platform Designer interconnect supports the following implementation scenarios:

- Any number of components with master and slave interfaces. The master-to-slave relationship can be one-to-one, one-to-many, many-to-one, or many-to-many.
- Masters and slaves of different data widths.
- Masters and slaves operating in different clock domains.
- IP Components with different interface properties and signals. Platform Designer adapts the component interfaces so that interfaces with the following differences can be connected:
  - Avalon and AXI interfaces that use active-high and active-low signaling. AXI signals are active high, except for the reset signal.
  - Interfaces with different burst characteristics.
  - Interfaces with different latencies.
  - Interfaces with different data widths.
  - Interfaces with different optional interface signals.

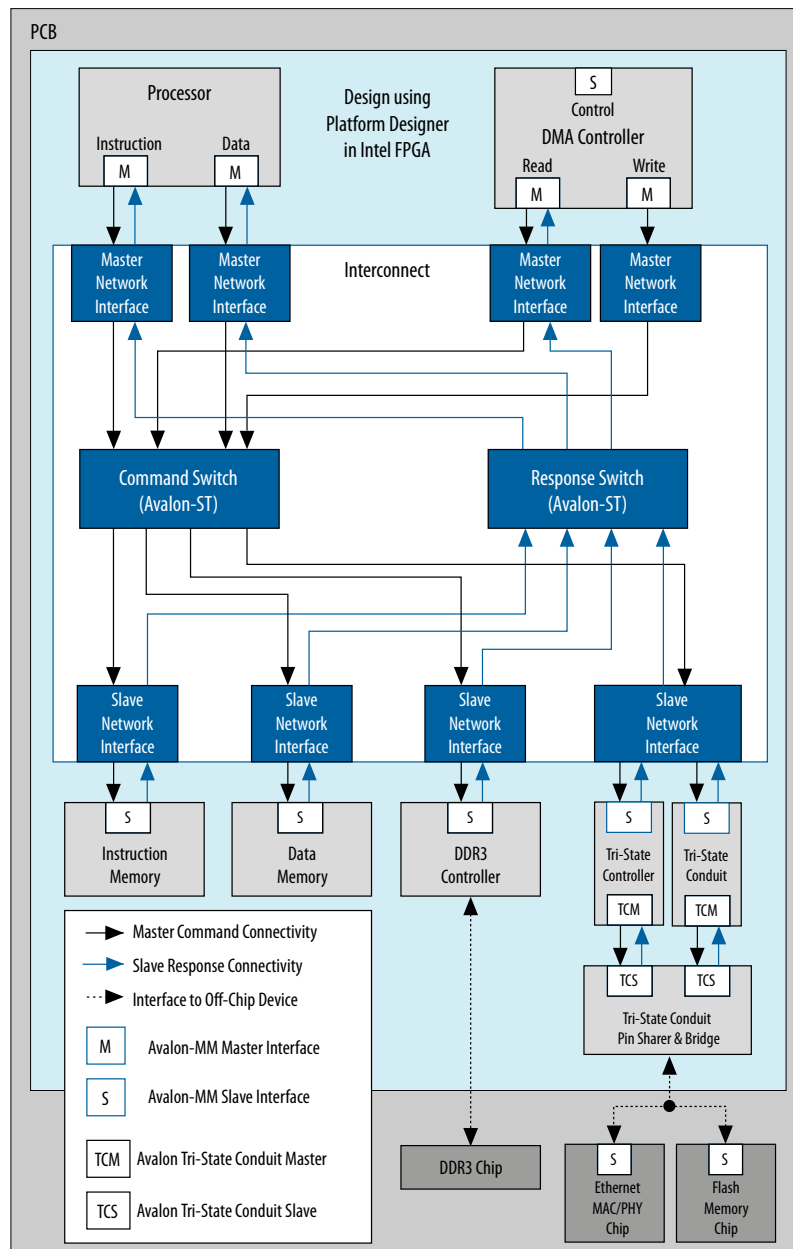
*Note:* Since interface connections between AMBA 3 AXI and AMBA 4 AXI declare a fixed set of signals with variable latency, there is no need for adapting between active-low and active-high signaling, burst characteristics, different latencies, or port signatures. Adaptation might be necessary between Avalon interfaces.

In this example, there are two components mastering the system, a processor and a DMA controller, each with two master interfaces. The masters connect through the Platform Designer interconnect to slaves in the Platform Designer system.

The dark blue blocks represent interconnect components. The dark grey boxes indicate items outside of the Platform Designer system and the Intel Quartus Prime software design, and show how to export component interfaces and how to connect these interfaces to external devices.



Figure 211. Platform Designer interconnect for an Avalon-MM System with Multiple Masters



### 11.1.1 Platform Designer Packet Format

The Platform Designer packet format supports Avalon, AXI, and APB transactions. Memory-mapped transactions between masters and slaves are encapsulated in Platform Designer packets. For Avalon systems without AXI or APB interfaces, some fields are ignored or removed.



### 11.1.1.1 Fields in the Platform Designer Packet Format

The fields of the Platform Designer packet format are of variable length to minimize resource usage. However, if most components in a design have a single data width, for example 32-bits, and a single component has a data width of 64-bits, Platform Designer inserts a width adapter to accommodate 64-bit transfers.

**Table 127. Platform Designer Packet Format for Memory-Mapped Master and Slave Interfaces**

Command	Description
Address	Specifies the byte address for the lowest byte in the current cycle. There are no restrictions on address alignment.
Size	Encodes the run-time size of the transaction. In conjunction with address, this field describes the segment of the payload that contains valid data for a beat within the packet.
Address Sideband	Carries "address" sideband signals. The interconnect passes this field from master to slave. This field is valid for each beat in a packet, even though it is only produced and consumed by an address cycle. Up to 8-bit sideband signals are supported for both read and write address channels.
Cache	Carries the AXI cache signals.
Transaction (Exclusive)	Indicates whether the transaction has exclusive access.
Transaction (Posted)	Used to indicate non-posted writes (writes that require responses).
Data	For command packets, carries the data to be written. For read response packets, carries the data that has been read.
Byteenable	Specifies which symbols are valid. AXI can issue or accept any byteenable pattern. For compatibility with Avalon, Intel recommends that you use the following legal values for 32-bit data transactions between Avalon masters and slaves: <ul style="list-style-type: none"> <li>• <b>1111</b>—Writes full 32 bits</li> <li>• <b>0011</b>—Writes lower 2 bytes</li> <li>• <b>1100</b>—Writes upper 2 bytes</li> <li>• <b>0001</b>—Writes byte 0 only</li> <li>• <b>0010</b>—Writes byte 1 only</li> <li>• <b>0100</b>—Writes byte 2 only</li> <li>• <b>1000</b>—Writes byte 3 only</li> </ul>
Source_ID	The ID of the master or slave that initiated the command or response.
Destination_ID	The ID of the master or slave to which the command or response is directed.
Response	Carries the AXI response signals.
Thread ID	Carries the AXI transaction ID values.
Byte count	The number of bytes remaining in the transaction, including this beat. Number of bytes requested by the packet.
<i>continued...</i>	



Command	Description
Burstwrap	<p>The burstwrap value specifies the wrapping behavior of the current burst. The burstwrap value is of the form <math>2^{&lt;n&gt;} - 1</math>. The following types are defined:</p> <ul style="list-style-type: none"> <li>Variable wrap—Variable wrap bursts can wrap at any integer power of 2 value. When the burst reaches the wrap boundary, it wraps back to the previous burst boundary so that only the low order bits are used for addressing. For example, a burst starting at address 0x1C, with a burst wrap boundary of 32 bytes and a burst size of 20 bytes, would write to addresses 0x1C, 0x0, 0x4, 0x8, and 0xC.</li> <li>For a burst wrap boundary of size <math>&lt;m&gt;</math>, <math>\text{Burstwrap} = &lt;m&gt; - 1</math>, or for this case <math>\text{Burstwrap} = (32 - 1) = 31</math> which is <math>2^5 - 1</math>.</li> <li>For AXI masters, the burstwrap boundary value (m) is based on the different AXBURST: <ul style="list-style-type: none"> <li>Burstwrap set to all 1's. For example, for a 6-bit burstwrap, burstwrap is 6'b111111.</li> <li>For WRAP bursts, <math>\text{burstwrap} = \text{AXLEN} * \text{size} - 1</math>.</li> <li>For FIXED bursts, <math>\text{burstwrap} = \text{size} - 1</math>.</li> <li>Sequential bursts increment the address for each transfer in the burst. For sequential bursts, the <math>\text{Burstwrap}</math> field is set to all 1s. For example, with a 6-bit <math>\text{Burstwrap}</math> field, the value for a sequential burst is 6'b111111 or 63, which is <math>2^6 - 1</math>.</li> </ul> </li> </ul> <p>For Avalon masters, Platform Designer adaptation logic sets a hardwired value for the burstwrap field, according the declared master burst properties. For example, for a master that declares sequential bursting, the burstwrap field is set to ones. Similarly, masters that declare burst have their burstwrap field set to the appropriate constant value.</p> <p>AXI masters choose their burst type at run-time, depending on the value of the AW or ARBURST signal. The interconnect calculates the burstwrap value at run-time for AXI masters.</p>
Protection	<p>Access level protection. When the lowest bit is 0, the packet has normal access. When the lowest bit is 1, the packet has privileged access. For Avalon-MM interfaces, this field maps directly to the privileged access signal, which allows a memory-mapped master to write to an on-chip memory ROM instance. The other bits in this field support AXI secure accesses and uses the same encoding, as described in the AXI specification.</p>
QoS	<p>QoS (Quality of Service Signaling) is a 4-bit field that is part of the AMBA 4 AXI interface that carries QoS information for the packet from the AXI master to the AXI slave. Transactions from AMBA 3 AXI and Avalon masters have the default value 4'b0000, that indicates that they are not participating in the QoS scheme. QoS values are dropped for slaves that do not support QoS.</p>
Data sideband	<p>Carries data sideband signals for the packet. On a write command, the data sideband directly maps to WUSER. On a read response, the data sideband directly maps to RUSER. On a write response, the data sideband directly maps to BUSER.</p>

### 11.1.1.2 Transaction Types for Memory-Mapped Interfaces

**Table 128. Transaction Types for Memory-Mapped Interfaces**

The table below describes the information that each bit transports in the packet format's transaction field.

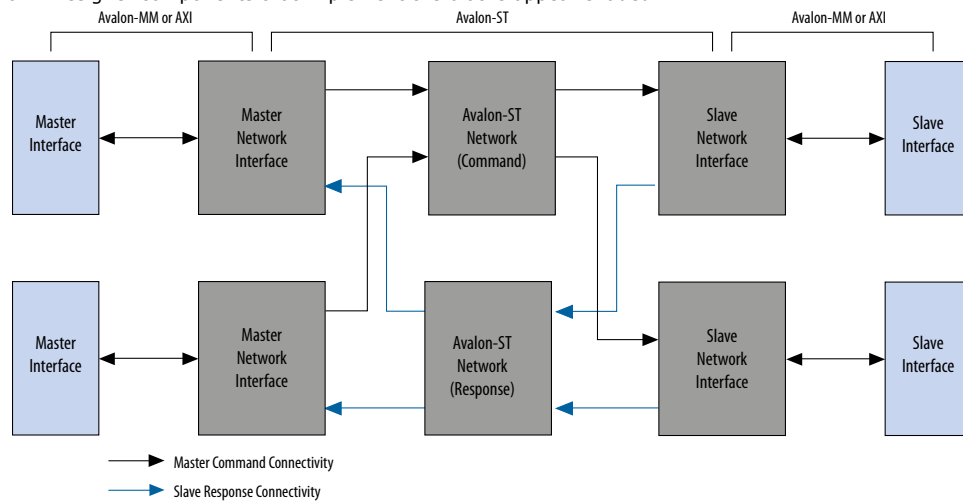
Bit	Name	Definition
0	PKT_TRANS_READ	When asserted, indicates a read transaction.
1	PKT_TRANS_COMPRESSED_READ	For read transactions, specifies whether the read command can be expressed in a single cycle (all <code>byteenables</code> asserted on every cycle).
2	PKT_TRANS_WRITE	When asserted, indicates a write transaction.
3	PKT_TRANS_POSTED	When asserted, no response is required.
4	PKT_TRANS_LOCK	When asserted, indicates arbitration is locked. Applies to write packets.

### 11.1.1.3 Platform Designer Transformations

The memory-mapped master and slave components connect to network interface modules that encapsulate the transaction in Avalon-ST packets. The memory-mapped interfaces have no information about the encapsulation or the function of the layer transporting the packets. The interfaces operate in accordance with memory-mapped protocol and use the read and write signals and transfers.

**Figure 212. Transformation when Generating a System with Memory-Mapped and Slave Components**

Platform Designer components that implement the blocks appear shaded.



#### Related Links

- [Master Network Interfaces](#) on page 666
- [Slave Network Interfaces](#) on page 669

### 11.1.2 Interconnect Domains

An interconnect domain is a group of connected memory-mapped masters and slaves that share the same interconnect. The components in a single interconnect domain share the same packet format.

#### 11.1.2.1 Using One Domain with Width Adaptation

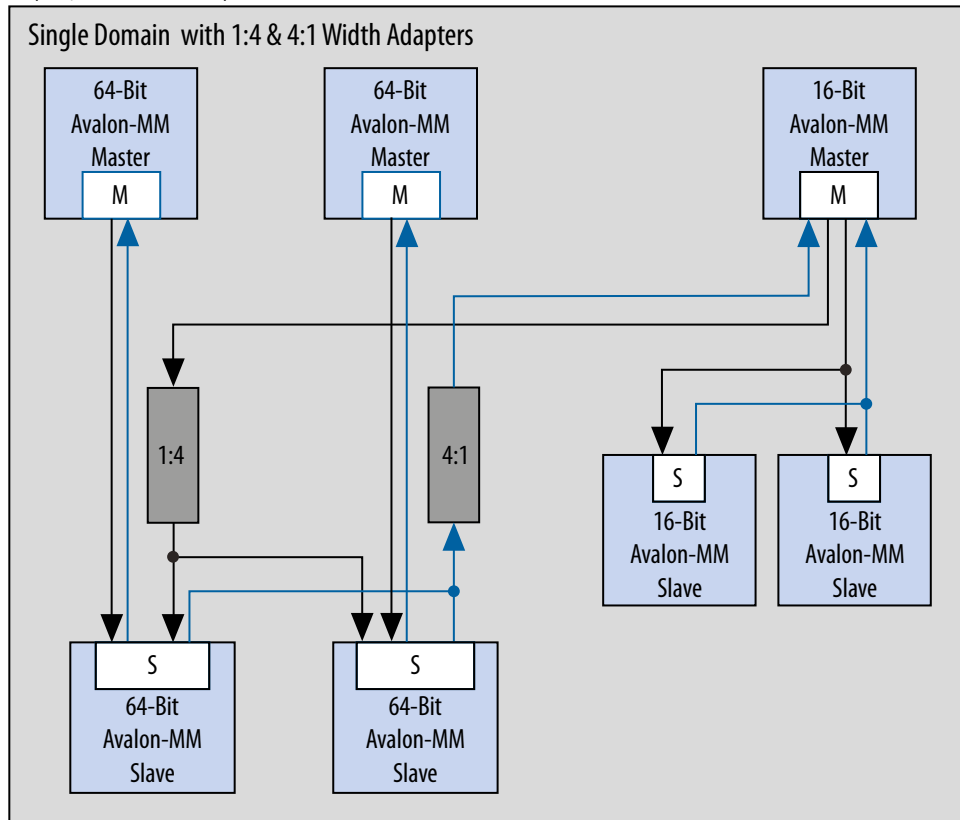
When one of the masters in a system connects to all the slaves, Platform Designer creates a single domain with two packet formats: one with 64-bit data, and one with 16-bit data. A width adapter manages accesses between the 16-bit master and 64-bit slaves.





**Figure 213. One Domain with 1:4 and 4:1 Width Adapters**

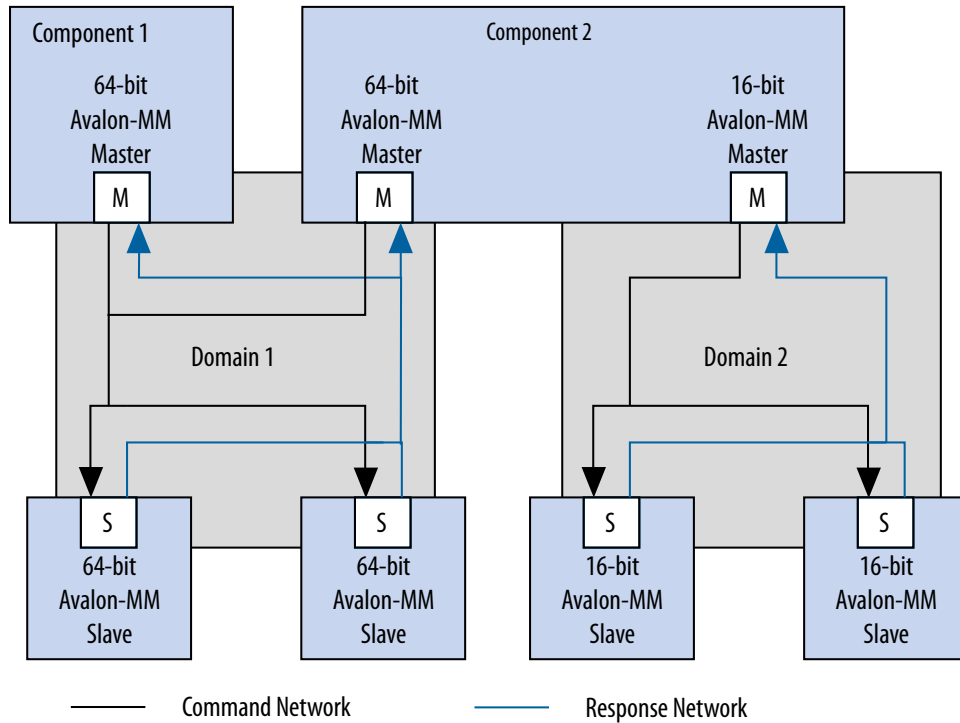
In this system example, there are two 64-bit masters that access two 64-bit slaves. It also includes one 16-bit master, that accesses two 16-bit slaves and two 64-bit slaves. The 16-bit Avalon master connects through a 1:4 adapter, then a 4:1 adapter to reach its 16-bit slaves.



### 11.1.2.2 Using Two Separate Domains

**Figure 214. Two Separate Domains**

In this system example, Platform Designer uses two separate domains. The first domain includes two 64-bit masters connected to two 64-bit slaves. A second domain includes one 16-bit master connected to two 16-bit slaves. Because the interfaces in Domain 1 and Domain 2 do not share any connections, Platform Designer can optimize the packet format for the two separate domains. In this example, the first domain uses a 64-bit data width and the second domain uses 16-bit data.

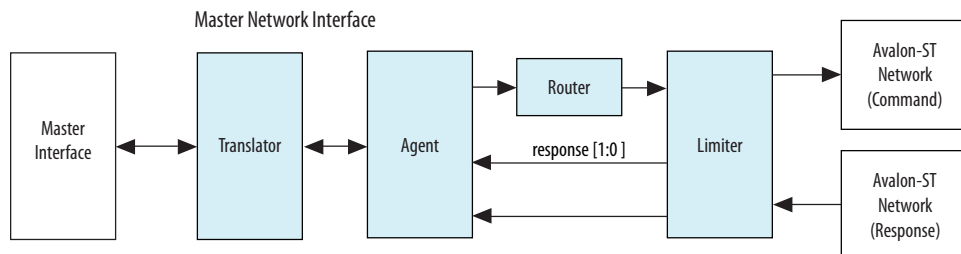


### 11.1.3 Master Network Interfaces

**Figure 215. Avalon-MM Master Network Interface**

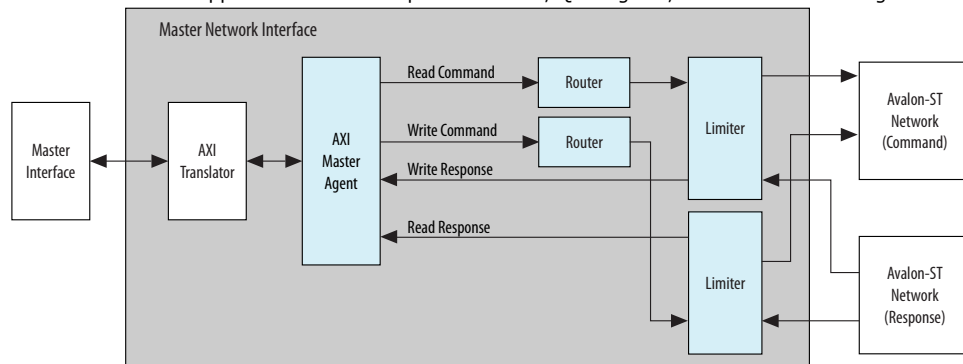
Avalon network interfaces drive default values for the QoS and BUSER, WUSER, and RUSER packet fields in the master agent, and drop the packet fields in the slave agent.

*Note:* The response signal from the Limiter to the Agent is optional.



**Figure 216. AXI Master Network Interface**

An AMBA 4 AXI master supports INCR bursts up to 256 beats, QoS signals, and data sideband signals.



*Note:* For a complete definition of the optional read response signal, refer to *Avalon Memory-Mapped Interface Signal Types* in the *Avalon Interface Specifications*.

#### Related Links

- [Avalon Interface Specifications](#)
- [Creating a System with Platform Designer](#) on page 327

#### 11.1.3.1 Avalon-MM Master Agent

The Avalon-MM Master Agent translates Avalon-MM master transactions into Platform Designer command packets and translates the Platform Designer Avalon-MM slave response packets into Avalon-MM responses.

#### 11.1.3.2 Avalon-MM Master Translator

The Avalon-MM Master Translator interfaces with an Avalon-MM master component and converts the Avalon-MM master interface to a simpler representation for use in Platform Designer.

The Avalon-MM Master translator performs the following functions:

- Translates active-low signaling to active-high signaling
- Inserts wait states to prevent an Avalon-MM master from reading invalid data
- Translates word and symbol addresses
- Translates word and symbol burst counts
- Manages re-timing and re-sequencing bursts
- Removes unnecessary address bits

#### 11.1.3.3 AXI Master Agent

An AXI Master Agent accepts AXI commands and produces Platform Designer command packets. It also accepts Platform Designer response packets and converts those into AXI responses. This component has separate packet channels for read commands, write commands, read responses, and write responses. Avalon master agent drives the QoS and BUSER, WUSER, and RUSER packet fields with default values AXQ0 and b0000, respectively.



**Note:** For signal descriptions, refer to *Platform Designer Packet Format*.

#### Related Links

[Fields in the Platform Designer Packet Format](#) on page 662

### 11.1.3.4 AXI Translator

AMBA 4 AXI allows signals to be omitted from interfaces. The translator bridges between these “incomplete” AMBA 4 AXI interfaces and the “complete” AMBA 4 AXI interface on the network interfaces.

The AXI translator is inserted for both AMBA 4 AXI masters and slaves and performs the following functions:

- Matches ID widths between the master and slave in 1x1 systems.
- Drives default values as defined in the *AMBA Protocol Specifications* for missing signals.
- Performs lock transaction bit conversion when an AMBA 3 AXI master connects to an AMBA 4 AXI slave in 1x1 systems.

#### Related Links

[Arm AMBA Protocol Specifications](#)

### 11.1.3.5 APB Master Agent

An APB master agent accepts APB commands and produces or generates Platform Designer command packets. It also converts Platform Designer response packets to APB responses.

### 11.1.3.6 APB Slave Agent

An APB slave agent issues resulting transaction to the APB interface. It also accepts creates Platform Designer response packets.

### 11.1.3.7 APB Translator

An APB peripheral does not require `pslverr` signals to support additional signals for the APB debug interface.

The APB translator is inserted for both the master and slave and performs the following functions:

- Sets the response value default to `OKAY` if the APB slave does not have a `pslverr` signal.
- Turns on or off additional signals between the APB debug interface, which is used with HPS (Intel SoC’s Hard Processor System).

### 11.1.3.8 AHB Slave Agent

The Platform Designer interconnect supports non-bursting Advanced High-performance Bus (AHB) slave interfaces.



### 11.1.3.9 Memory-Mapped Router

The Memory-Mapped Router routes command packets from the master to the slave, and response packets from the slave to the master. For master command packets, the router uses the address to set the `Destination_ID` and Avalon-ST channel. For the slave response packet, the router uses the `Destination_ID` to set the Avalon-ST channel. The demultiplexers use the Avalon-ST channel to route the packet to the correct destination.

### 11.1.3.10 Memory-Mapped Traffic Limiter

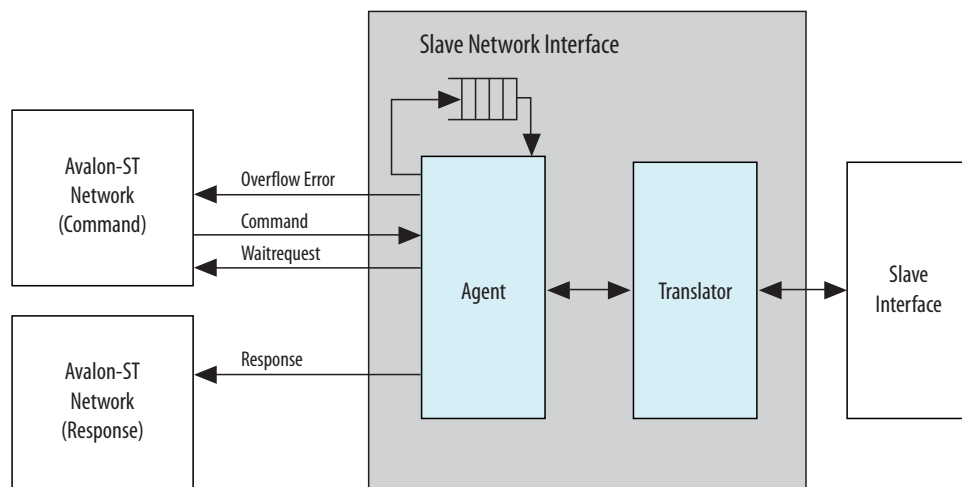
The Memory-Mapped Traffic Limiter ensures the responses arrive in order. It prevents any command from being sent if the response could conflict with the response for a command that has already been issued. By guaranteeing in-order responses, the Traffic Limiter simplifies the response network.

## 11.1.4 Slave Network Interfaces

### 11.1.4.1 Avalon-MM Slave Translator

The Avalon-MM Slave Translator converts the Avalon-MM slave interface to a simplified representation that the Platform Designer network can use.

Figure 217. Avalon-MM Slave Network Interface



An Avalon-MM Slave Translator performs the following functions:

- Drives the `beginbursttransfer` and `byteenable` signals.
- Supports Avalon-MM slaves that operate using fixed timing and or slaves that use the `readdatavalid` signal to identify valid data.
- Translates the `read`, `write`, and `chipselct` signals into the representation that the Avalon-ST slave response network uses.
- Converts active low signals to active high signals.

- Translates word and symbol addresses and burstcounts.
- Handles burstcount timing and sequencing.
- Removes unnecessary address bits.

### Related Links

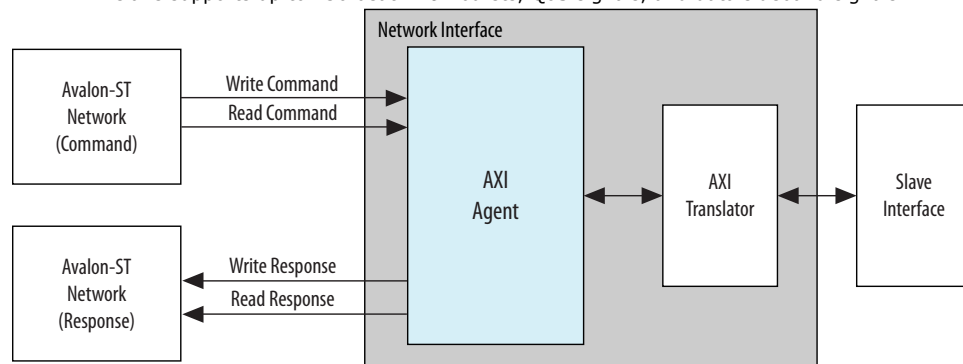
[Slave Network Interfaces](#) on page 669

#### 11.1.4.2 AXI Translator

AMBA 4 AXI allows omitting signals from interfaces. The translator bridges between these “incomplete” AMBA 4 AXI interfaces and the “complete” AMBA 4 AXI interface on the network interfaces.

#### Figure 218. AXI Slave Network Interface

An AMBA 4 AXI slave supports up to 256 beat INCR bursts, QoS signals, and data sideband signals.



The AXI translator is inserted for both AMBA 4 AXI master and slave, and performs the following functions:

- Matches ID widths between master and slave in 1x1 systems.
- Drives default values as defined in the *AMBA Protocol Specifications* for missing signals.
- Performs lock transaction bit conversion when an AMBA 3 AXI master connects to an AMBA 4 AXI slave in 1x1 systems.

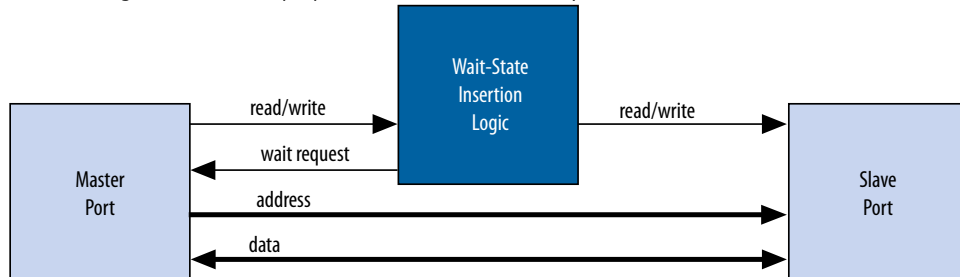
#### 11.1.4.3 Wait State Insertion

Wait states extend the duration of a transfer by one or more cycles. Wait state insertion logic accommodates the timing needs of each slave, and causes the master to wait until the slave can proceed. Platform Designer interconnect inserts wait states into a transfer when the target slave cannot respond in a single clock cycle, as well as in cases when slave `read` and `write` signals have setup or hold time requirements.



**Figure 219. Wait State Insertion Logic for One Master and One Slave**

Wait state insertion logic is a small finite-state machine that translates control signal sequencing between the slave side and the master side. Platform Designer interconnect can force a master to wait for the wait state needs of a slave; for example, arbitration logic in a multi-master system. Platform Designer generates wait state insertion logic based on the properties of all slaves in the system.



#### 11.1.4.4 Avalon-MM Slave Agent

The Avalon-MM Slave Agent accepts command packets and issues the resulting transactions to the Avalon interface. For pipelined slaves, an Avalon-ST FIFO stores information about pending transactions. The size of this FIFO is the maximum number of pending responses that you specify when creating the slave component. The Avalon-MM Slave Agent also backpressures the Avalon-MM master command interface when the FIFO is full if the slave component includes the `waitrequest` signal.

#### 11.1.4.5 AXI Slave Agent

An AXI Slave Agent works like a reverse master agent. The AXI Slave Agent accepts Platform Designer command packets to create AXI commands, and accepts AXI responses to create Platform Designer response packets. This component has separate packet channels for read commands, write commands, read responses, and write responses.

#### 11.1.5 Arbitration

When multiple masters contend for access to a slave, Platform Designer automatically inserts arbitration logic, which grants access in fairness-based, round-robin order. You can alternatively choose to designate a slave as a fixed priority arbitration slave, and then manually assign priorities in the Platform Designer GUI.

##### 11.1.5.1 Round-Robin Arbitration

When multiple masters contend for access to a slave, Platform Designer automatically inserts arbitration logic which grants access in fairness-based, round-robin order.

In a fairness-based arbitration protocol, each master has an integer value of transfer *shares* with respect to a slave. One share represents permission to perform one transfer. The default arbitration scheme is equal share round-robin that grants equal, sequential access to all requesting masters. You can change the arbitration scheme to weighted round-robin by specifying a relative number of arbitration shares to the masters that access a given slave. AXI slaves have separate arbitration for their independent read and write channels, and the **Arbitration Shares** setting affects both the read and write arbitration. To display arbitration settings, right-click an instance on the **System Contents** tab, and then click **Show Arbitration Shares**.

Figure 220. Arbitration Shares in the Connections Column

Connections	Name	Description
	<b>mm_master_bfm_0_avalon</b>	Altera UVM Avalon-MM Master BFM
	clk	Clock Input
	clk_reset	Reset Input
	m0	Avalon Memory Mapped Master
	<b>mm_master_bfm_1_axi</b>	Altera AXI3 Master Module
	clk	Clock Input
	clk_reset	Reset Input
	altera_axi_master	AXI Master
	<b>mm_master_bfm_2_axi</b>	Altera AXI3 Master Module
	clk	Clock Input
clk_reset	Reset Input	
altera_axi_master	AXI Master	
<b>mm_slave_bfm_0_avalon</b>	Altera UVM Avalon-MM Slave BFM	
clk	Clock Input	
clk_reset	Reset Input	
s0	Avalon Memory Mapped Slave	
<b>mm_slave_bfm_1_avalon</b>	Altera UVM Avalon-MM Slave BFM	
clk	Clock Input	
clk_reset	Reset Input	
s0	Avalon Memory Mapped Slave	
<b>mm_slave_bfm_2_axi</b>	Altera AXI3 Slave Module	
clk	Clock Input	
clk_reset	Reset Input	
altera_axi_slave	AXI Slave	
<b>CLOCK_0</b>	Altera Avalon Clock and Reset Source	
clk	Clock Output	
clk_reset	Reset Output	
dummy_src	Avalon Streaming Source	
dummy_snk	Avalon Streaming Sink	

### 11.1.5.1.1 Fairness-Based Shares

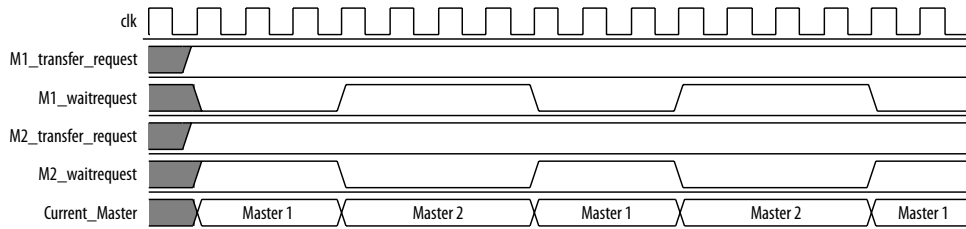
In a fairness-based arbitration scheme, each master-to-slave connection provides a transfer share count. This count is a request for the arbiter to grant a specific number of transfers to this master before giving control to a different master. One share represents permission to perform one transfer.





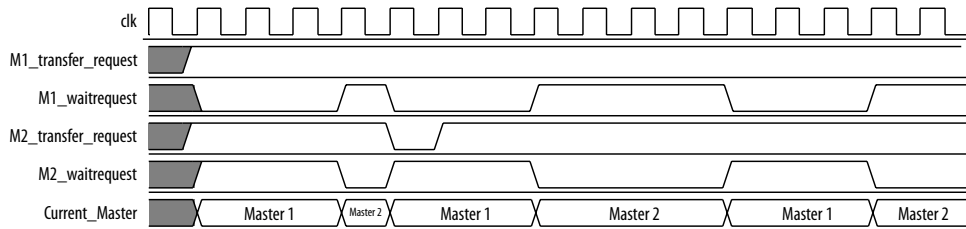
**Figure 221. Arbitration of Continuous Transfer Requests from Two Masters**

Consider a system with two masters connected to a single slave. Master 1 has its arbitration shares set to three, and Master 2 has its arbitration shares set to four. Master 1 and Master 2 continuously attempt to perform back-to-back transfers to the slave. The arbiter grants Master 1 access to the slave for three transfers, and then grants Master 2 access to the slave for four transfers. This cycle repeats indefinitely. The figure below describes the waveform for this scenario.



**Figure 222. Arbitration of Two Masters with a Gap in Transfer Requests**

If a master stops requesting transfers before it exhausts its shares, it forfeits all its remaining shares, and the arbiter grants access to another requesting master. After completing one transfer, Master 2 stops requesting for one clock cycle. As a result, the arbiter grants access back to Master 1, which gets a replenished supply of shares.



### 11.1.5.1.2 Round-Robin Scheduling

When multiple masters contend for access to a slave, the arbiter grants shares in round-robin order. Platform Designer includes only requesting masters in the arbitration for each slave transaction.

### 11.1.5.2 Fixed Priority Arbitration

Fixed priority arbitration is an alternative arbitration scheme to the default round-robin scheme.

You can selectively apply fixed priority arbitration to any slave in a Platform Designer system. You can design Platform Designer systems where a subset of slaves use the default round-robin arbitration, and other slaves use fixed priority arbitration. Fixed priority arbitration uses a fixed priority algorithm to grant access to a slave amongst its connected masters.

To set up fixed priority arbitration, you must first designate a fixed priority slave in your Platform Designer system in the **Interconnect Requirements** tab. You can then assign an arbitration priority number for each master connected to a fixed priority slave in the **System Contents** tab, where the highest numeric value receives the highest priority. When multiple masters request access to a fixed priority arbitrated slave, the arbiter gives the master with the highest priority first access to the slave.

For example, when a fixed priority slave receives requests from three masters on the same cycle, the arbiter grants the master with highest assigned priority first access to the slave, and backpressures the other two masters.



**Note:** When you connect an AXI master to an Avalon-MM slave designated to use a fixed priority arbitrator, the interconnect instantiates a command-path intermediary round-robin multiplexer in front of the designated slave.

#### 11.1.5.2.1 Designate a Platform Designer Slave to Use Fixed Priority Arbitration

You can designate any slave in your Platform Designer system to use fixed priority arbitration. You must assign each master connected to a fixed priority slave a numeric priority. The master with the highest higher priority receives first access to the slave. No two masters can have the same priority.

1. In Platform Designer, navigate to the **Interconnect Requirements** tab.
2. Click **Add** to add a new requirement.
3. In the **Identifier** column, select the slave for fixed priority arbitration.
4. In the **Setting** column, select **qsys mm.arbitrationScheme**.
5. In the **Value** column, select **fixed-priority**.
6. Navigate to the **System Contents** tab.
7. In the **System Contents** tab, right-click the designated fixed priority slave, and then select **Show Arbitration Shares**.
8. For each master connected to the fixed priority arbitration slave, type a numerical arbitration priority in the box that appears in place of the connection circle.
9. Right click the designated fixed priority slave and uncheck **Show Arbitration Shares** to return to the connection circles.

#### 11.1.5.2.2 Fixed Priority Arbitration with AXI Masters and Avalon-MM Slaves

When an AXI master is connected to a designated fixed priority arbitration Avalon-MM slave, Platform Designer interconnect automatically instantiates an intermediary multiplexer in front of the Avalon-MM slave.

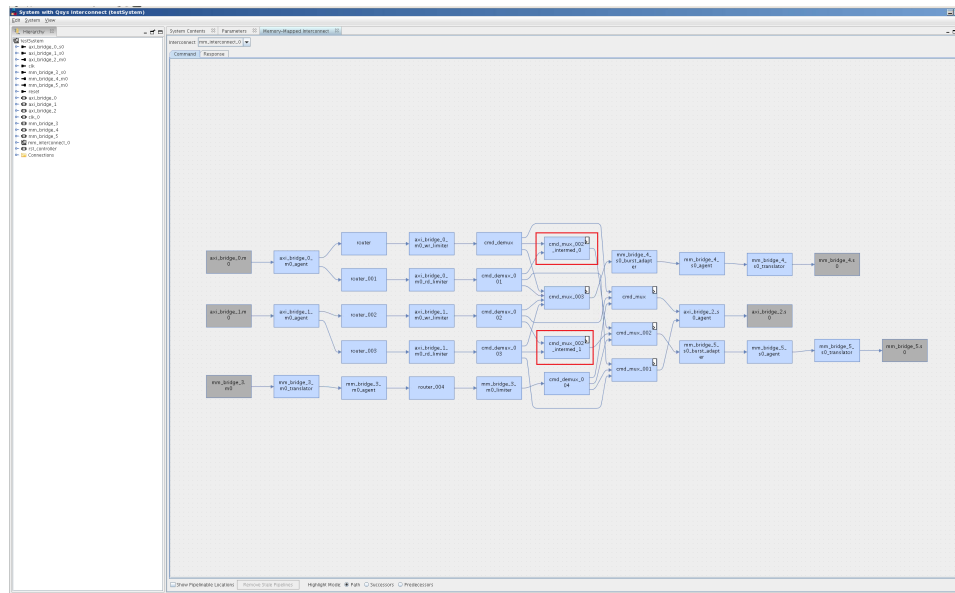
Since AXI masters have separate read and write channels, each channel appears as two separate masters to the Avalon-MM slave. To support fairness between the AXI master's read and write channels, the instantiated round-robin intermediary multiplexer arbitrates between simultaneous read and write commands from the AXI master to the fixed-priority Avalon-MM slave.

When an AXI master is connected to a fixed priority AXI slave, the master's read and write channels are directly connected to the AXI slave's fixed-priority multiplexers. In this case, there is one multiplexer for the read command, and one multiplexer for the write command and therefore an intermediary multiplexer is not required.

The red circles indicate placement of the intermediary multiplexer between the AXI master and Avalon-MM slave due to the separate read and write channels of the AXI master.



Figure 223. Intermediary Multiplexer Between AXI Master and Avalon-MM Slave

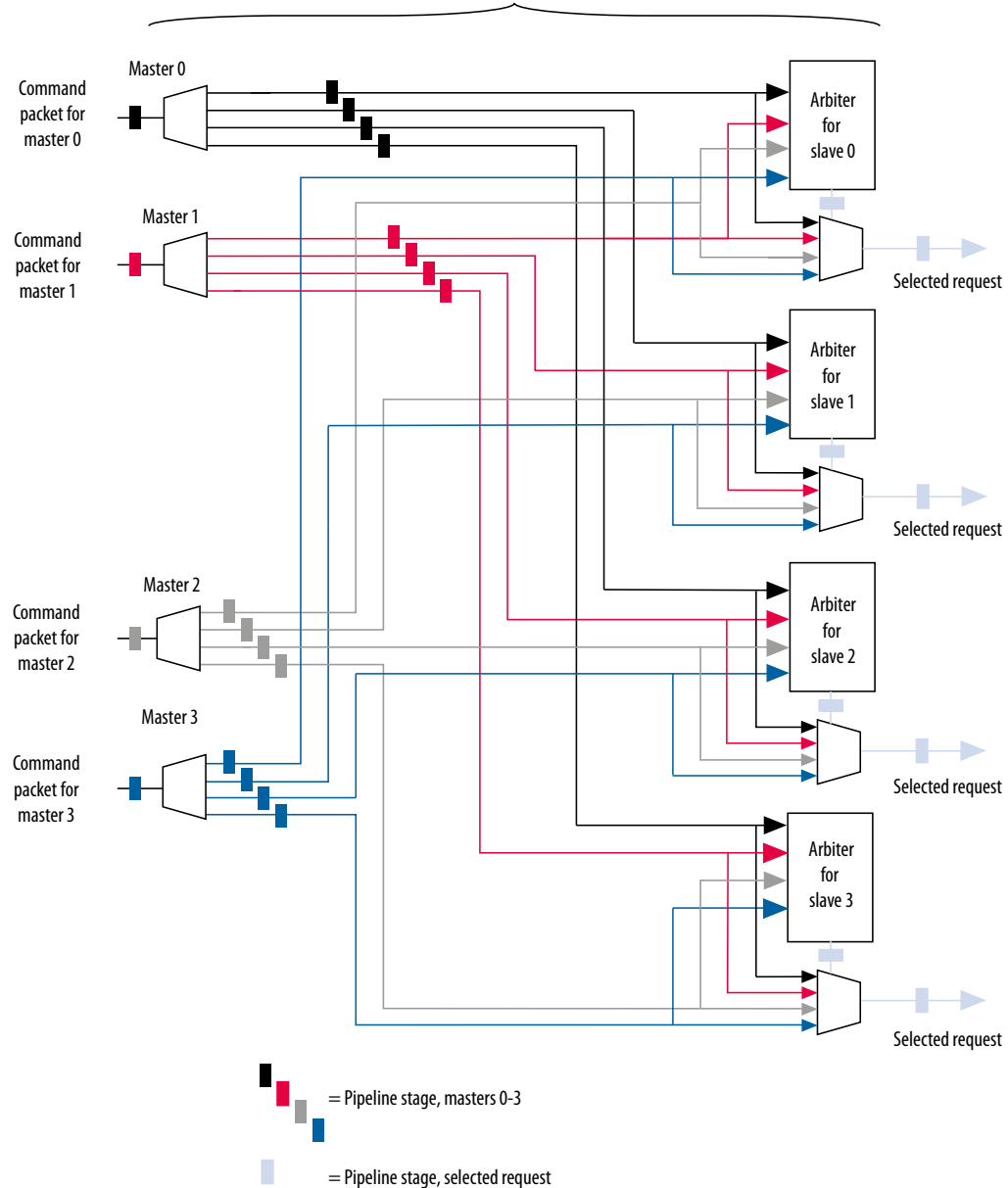


### 11.1.6 Memory-Mapped Arbiter

The input to the Memory-Mapped Arbiter is the command packet for all masters requesting access to a specific slave. The arbiter outputs the channel number for the selected master. This channel number controls the output of a multiplexer that selects the slave device. The figure below illustrates this logic.

**Figure 224. Arbitration Logic**

In this example, four Avalon-MM masters connect to four Avalon-MM slaves. In each cycle, an arbiter positioned in front of each Avalon-MM slave selects among the requesting Avalon-MM masters.  
Logic included in the Avalon-ST Command Network



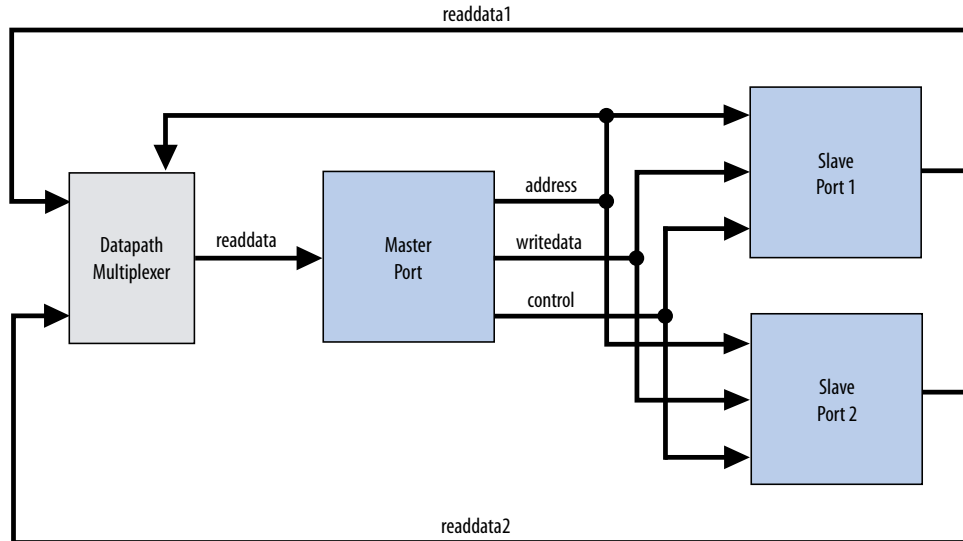
**Note:** If you specify a **Limit interconnect pipeline stages to** parameter greater than zero, the output of the Arbiter is registered. Registering this output reduces the amount of combinational logic between the master and the interconnect, increasing the  $f_{MAX}$  of the system.

**Note:** You can use the Memory-Mapped Arbiter for both round-robin and fixed priority arbitration.

### 11.1.7 Datapath Multiplexing Logic

Datapath multiplexing logic drives the `writedata` signal from the granted master to the selected slave, and the `readdata` signal from the selected slave back to the requesting master. Platform Designer generates separate datapath multiplexing logic for every master in the system (`readdata`), and for every slave in the system (`writedata`). Platform Designer does not generate multiplexing logic if it is not needed.

Figure 225. Datapath Multiplexing Logic for One Master and Two Slaves



### 11.1.8 Width Adaptation

Platform Designer width adaptation converts between Avalon memory-mapped master and slaves with different data and byte enable widths, and manages the run-time size requirements of AXI. Width adaptation for AXI to Avalon interfaces is also supported.

#### 11.1.8.1 Memory-Mapped Width Adapter

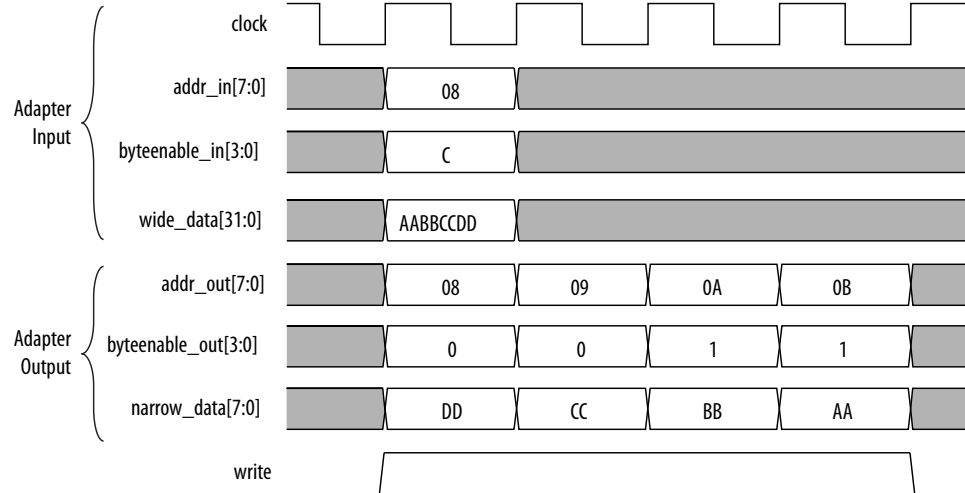
The Memory-Mapped Width Adapter is used in the Avalon-ST domain and operates with information contained in the packet format.

The memory-mapped width adapter accepts packets on its sink interface with one data width and produces output packets on its source interface with a different data width. The ratio of the narrow data width must be a power of two, such as 1:4, 1:8, and 1:16. The ratio of the wider data width to the narrower width must also be a power of two, such as 4:1, 8:1, and 16:1. These output packets may have a different size if the input size exceeds the output data bus width, or if data packing is enabled.

When the width adapter converts from narrow data to wide data, each input beat's data and byte enables are copied to the appropriate segment of the wider output data and byte enables signals.

**Figure 226. Width Adapter Timing for a 4:1 Adapter**

This adapter assumes that the field ordering of the input and output packets is the same, with the only difference being the width of the data and accompanying byte enable fields. When the width adapter converts from wide data to narrow data, the narrower data is transmitted over several beats. The first output beat contains the lowest addressed segment of the input data and byte enables.



#### 11.1.8.1.1 AXI Wide-to-Narrow Adaptation

For all cases of AXI wide-to-narrow adaptation, read data is re-packed to match the original size. Responses are merged, with the following error precedence: `DECERR`, `SLVERR`, `OKAY`, and `EXOKAY`.

**Table 129. AXI Wide-to-Narrow Adaptation (Downsizing)**

Burst Type	Behavior
Incrementing	<p>If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted to an incrementing burst with a larger length and size equal to the output width.</p> <p>If the resulting burst is unsuitable for the slave, the burst is converted to multiple sequential bursts of the largest allowable lengths. For example, for a 2:1 downsizing ratio, an <code>INCR9</code> burst is converted into <code>INCR16 + INCR2</code> bursts. This is true if the maximum burstcount a slave can accept is 16, which is the case for AMBA 3 AXI slaves. Avalon slaves have a maximum burstcount of 64.</p>
Wrapping	<p>If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted to a wrapping burst with a larger length, with a size equal to the output width.</p> <p>If the resulting burst is unsuitable for the slave, the burst is converted to multiple sequential bursts of the largest allowable lengths; respecting wrap boundaries. For example, for a 2:1 downsizing ratio, a <code>WRAP16</code> burst is converted into two or three <code>INCR</code> bursts, depending on the address.</p>
Fixed	<p>If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted into repeated sequential bursts over the same addresses. For example, for a 2:1 downsizing ratio, a <code>FIXED</code> single burst is converted into an <code>INCR2</code> burst.</p>



### 11.1.8.1.2 AXI Narrow-to-Wide Adaptation

**Table 130. AXI Narrow-to-Wide Adaptation (Upsizing)**

Burst Type	Behavior
Incrementing	The burst (and its response) passes through unmodified. Data and write strobes are placed in the correct output segment.
Wrapping	The burst (and its response) passes through unmodified.
Fixed	The burst (and its response) passes through unmodified.

### 11.1.9 Burst Adapter

Platform Designer interconnect uses the memory-mapped burst adapter to accommodate the burst capabilities of each interface in the system, including interfaces that do not support burst transfers.

The maximum burst length for each interface is a property of the interface and is independent of other interfaces in the system. Therefore, a specific master may be capable of initiating a burst longer than a slave's maximum supported burst length. In this case, the burst adapter translates the large master burst into smaller bursts, or into individual slave transfers if the slave does not support bursting. Until the master completes the burst, arbiter logic prevents other masters from accessing the target slave. For example, if a master initiates a burst of 16 transfers to a slave with maximum burst length of 8, the burst adapter initiates 2 bursts of length 8 to the slave.

Avalon-MM and AXI burst transactions allow a master uninterrupted access to a slave for a specified number of transfers. The master specifies the number of transfers when it initiates the burst. Once a burst begins between a master and slave, arbiter logic is locked until the burst completes. For burst masters, the length of the burst is the number of cycles that the master has access to the slave, and the selected arbitration shares have no effect.

*Note:* AXI masters can issue burst types that Avalon cannot accept, for example, fixed bursts. In this case, the burst adapter converts the fixed burst into a sequence of transactions to the same address.

*Note:* For AMBA 4 AXI slaves, Platform Designer allows 256-beat INCR bursts. You must ensure that 256-beat narrow-sized INCR bursts are shortened to 16-beat narrow-sized INCR bursts for AMBA 3 AXI slaves.

Avalon-MM masters always issue addresses that are aligned to the size of the transfer. However, when Platform Designer uses a narrow-to-wide width adaptation, the resulting address may be unaligned. For unaligned addresses, the burst adapter issues the maximum sized bursts with appropriate byte enables. This brings the burst-in-progress up to an aligned slave address. Then, it completes the burst on aligned addresses.

The burst adapter supports variable wrap or sequential burst types to accommodate different properties of memory-mapped masters. Some bursting masters can issue more than one burst type.

Burst adaptation is available for Avalon to Avalon, Avalon to AXI, and AXI to Avalon, and AXI to AXI connections. For information about AXI-to-AXI adaptation, refer to *AXI Wide-to-Narrow Adaptation*



**Note:** For AMBA 4 AXI to AMBA 3 AXI connections, Platform Designer follows an AMBA 4 AXI 256 burst length to AMBA 3 AXI 16 burst length.

### 11.1.9.1 Burst Adapter Implementation Options

Platform Designer automatically inserts burst adapters into your system depending on your master and slave connections, and properties. You can select burst adapter implementation options on the **Interconnect Requirements** tab.

To access the implementation options, you must select the **Burst adapter implementation** setting for the \$system identifier.

- **Generic converter (slower, lower area)**—Default. Controls all burst conversions with a single converter that can adapt incoming burst types. This results in an adapter that has lower  $f_{MAX}$ , but smaller area.
- **Per-burst-type converter (faster, higher area)**—Controls incoming bursts with a specific converter, depending on the burst type. This results in an adapter that has higher  $f_{MAX}$ , but higher area. This setting is useful when you have AXI masters or slaves and you want a higher  $f_{MAX}$ .

**Note:** For more information about the **Interconnect Requirements** tab, refer to *Creating a System with Platform Designer*.

#### Related Links

[Creating a System with Platform Designer](#) on page 327

### 11.1.9.2 Burst Adaptation: AXI to Avalon

**Table 131. Burst Adaptation: AXI to Avalon**

Entries specify the behavior when converting between AXI and Avalon burst types.

Burst Type	Behavior
Incrementing	<p><b>Sequential Slave</b> Bursts that exceed <code>slave_max_burst_length</code> are converted to multiple sequential bursts of a length less than or equal to the <code>slave_max_burst_length</code>. Otherwise, the burst is unconverted. For example, for an Avalon slave with a maximum burst length of 4, an <code>INCR7</code> burst is converted to <code>INCR4 + INCR3</code>.</p> <p><b>Wrapping Slave</b> Bursts that exceed the <code>slave_max_burst_length</code> are converted to multiple sequential bursts of length less than or equal to the <code>slave_max_burst_length</code>. Bursts that exceed the wrapping boundary are converted to multiple sequential bursts that respect the slave's wrapping boundary.</p>
Wrapping	<p><b>Sequential Slave</b> A WRAP burst is converted to multiple sequential bursts. The sequential bursts are less than or equal to the <code>max_burst_length</code> and respect the transaction's wrapping boundary</p> <p><b>Wrapping Slave</b> If the WRAP transaction's boundary matches the slave's boundary, then the burst passes through. Otherwise, the burst is converted to sequential bursts that respect both the transaction and slave wrap boundaries.</p>
Fixed	Fixed bursts are converted to sequential bursts of length 1 that repeatedly access the same address.
Narrow	All narrow-sized bursts are broken into multiple bursts of length 1.





### 11.1.9.3 Burst Adaptation: Avalon to AXI

**Table 132. Burst Adaptation: Avalon to AXI**

Entries specify the behavior when converting between Avalon and AXI burst types.

Burst Type	Definition
Sequential	Bursts of length greater than 16 are converted to multiple INCR bursts of a length less than or equal to 16. Bursts of length less than or equal to 16 are not converted.
Wrapping	Only Avalon masters with <code>alwaysBurstMaxBurst = true</code> are supported. The WRAP burst is passed through if the length is less than or equal to 16. Otherwise, it is converted to two or more INCR bursts that respect the transaction's wrap boundary.
GENERIC_CONVERTER	Controls all burst conversions with a single converter that adapts all incoming burst types, resulting in an adapter that has smaller area, but lower $f_{MAX}$ .

### 11.1.10 Read and Write Responses

Platform Designer merges write responses if a write is converted (burst adapted) into multiple bursts. Platform Designer requires read response merging for a downsized (wide-to-narrow width adapted) read.

Platform Designer merges responses based on the following precedence rule:

```
DECERR > SLVERR > OKAY > EXOKAY
```

Adaptation between a master with write responses and a slave without write responses can be costly, especially if there are multiple slaves, or if the slave supports bursts. To minimize the cost of logic between slaves, consider placing the slaves that do not have write responses behind a bridge so that the write response adaptation logic cost is only incurred once, at the bridge's slave interface.

The following table describes what happens when there is a mismatch in response support between the master and slave.

**Table 133. Response Support for Mismatched Master and Slave**

	Slave with Response	Slave Without Response
Master with Response	Interconnect delivers response from the slave to the master. Response merging or duplication may be necessary for bus sizing.	Interconnect delivers an OKAY response to the master
Master without Response	Master ignores responses from the slave	No need for responses. Master, slave and interconnect operate without response support.

*Note:* If there is a bridge between the master and the endpoint slave, and the responses must come from the endpoint slave, ensure that the bridge passes the appropriate response signals through from the endpoint slave to the master.

If the bridge does not support responses, then the responses are generated by the interconnect at the slave interface of the bridge, and responses from the endpoint slave are ignored.

For the response case where the transaction violates security settings or uses an illegal address, the interconnect routes the transactions to the default slave. For information about Platform Designer system security, refer to Manage System Security. For information about specifying a default slave, refer to *Error Response Slave* in *Platform Designer System Design Components*.

**Note:** Avalon-MM slaves without a `response` signal are not able to notify a connected master that a transaction has not completed successfully. As a result, Platform Designer interconnect generates an `OKAY` response on behalf of the Avalon-MM slave.

### Related Links

- [Master Network Interfaces](#) on page 666
- [Error Response Slave](#) on page 937

## 11.1.11 Platform Designer Address Decoding

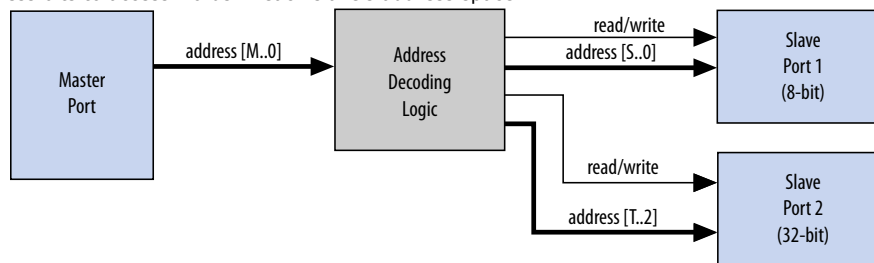
Address decoding logic forwards appropriate addresses to each slave.

Address decoding logic simplifies component design in the following ways:

- The interconnect selects a slave whenever it is being addressed by a master. Slave components do not need to decode the address to determine when they are selected.
- Slave addresses are properly aligned to the slave interface.
- Changing the system memory map does not involve manually editing HDL.

### Figure 227. Address Decoding for One Master and Two Slaves

In this example, Platform Designer generates separate address decoding logic for each master in a system. The address decoding logic processes the difference between the master address width ( $\langle M \rangle$ ) and the individual slave address widths ( $\langle S \rangle$ ) and ( $\langle T \rangle$ ). The address decoding logic also maps only the necessary master address bits to access words in each slave's address space.





**Figure 228. Address Decoding Base Settings**

Platform Designer controls the base addresses with the **Base** setting of active components on the **System Contents** tab. The base address of a slave component must be a multiple of the address span of the component. This restriction is part of the Platform Designer interconnect to allow the address decoding logic to be efficient, and to achieve the best possible  $f_{MAX}$ .

Name	Description	Export	Clock	Base	End	IRQ
<b>mem_clk</b>	Clock Source					
clk_in	Clock Input					
clk_in_reset	Reset Input					
clk	Clock Output		mem_clk			
clk_reset	Reset Output					
<b>cpu_clk</b>	Clock Source					
clk_in	Clock Input					
clk_in_reset	Reset Input					
clk	Clock Output		cpu_clk			
clk_reset	Reset Output					
<b>cpu</b>	Nios II Processor					
clk	Clock Input		cpu_clk			
reset_n	Reset Input					
data_master	Avalon Memory Mapped Master			IRQ 0	IRQ 31	
instruction_master	Avalon Memory Mapped Master					
jtag_debug_module_reset	Reset Output					
jtag_debug_module	Avalon Memory Mapped Slave			0x0001_2000	0x0001_27ff	
custom_instruction_master	Custom Instruction Master					
<b>onchip_ram</b>	On-Chip Memory (RAM or ROM)					
clk1	Clock Input		cpu_clk			
<b>s1</b>	Avalon Memory Mapped Slave			0x0001_0000	0x0001_1fff	
reset1	Reset Input					
<b>jtag_uart</b>	JTAG UART					
clk	Clock Input		cpu_clk			
reset	Reset Input					
avalon_jtag_slave	Avalon Memory Mapped Slave			0x0001_2800	0x0001_2807	
<b>pipeline_bridge</b>	Avalon-MM Pipeline Bridge					
clk	Clock Input		mem_clk			
reset	Reset Input					
s0	Avalon Memory Mapped Slave			0x0000_0000	0x0000_ffff	
m0	Avalon Memory Mapped Master	master				

## 11.2 Avalon Streaming Interfaces

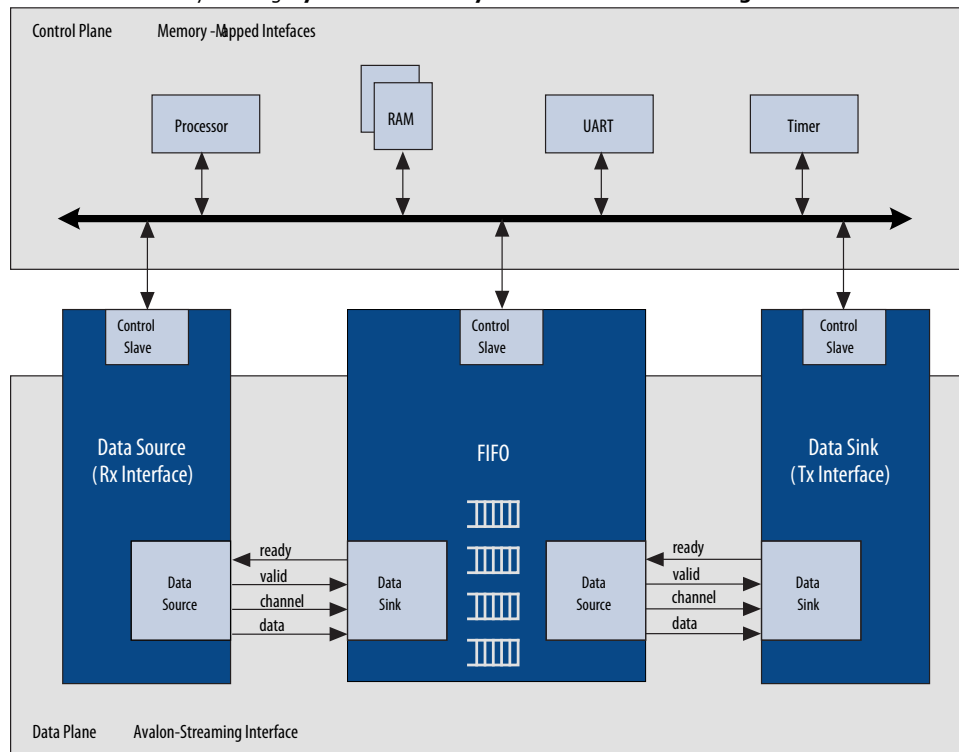
High bandwidth components with streaming data typically use Avalon-ST interfaces for the high throughput datapath. Streaming interfaces can also use memory-mapped connection interfaces to provide an access point for control. In contrast to the memory-mapped interconnect, the Avalon-ST interconnect always creates a point-to-point connection between a single data source and data sink.

**Figure 229. Memory-Mapped and Avalon-ST Interfaces**

In this example, there are the following connection pairs:

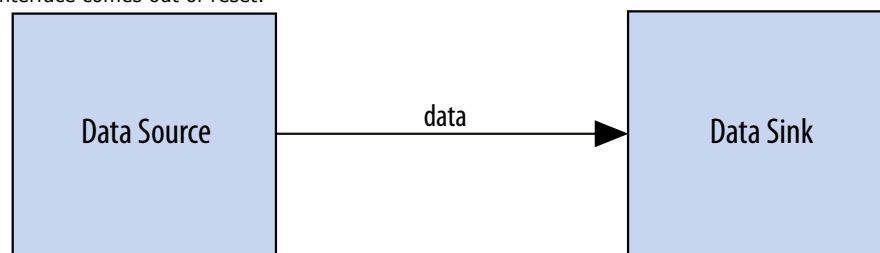
- Data source in the Rx Interface transfers data to the data sink in the FIFO.
- Data source in the FIFO transfers data to the Tx Interface data sink.

The memory-mapped interface allows a processor to access the data source, FIFO, or data sink to provide system control. If your source and sink interfaces have different formats, for example, a 32-bit source and an 8-bit sink, Platform Designer automatically inserts the necessary adapters. You can view the adapters on the **System Contents** tab by clicking **System > Show System with Platform Designer Interconnect**.



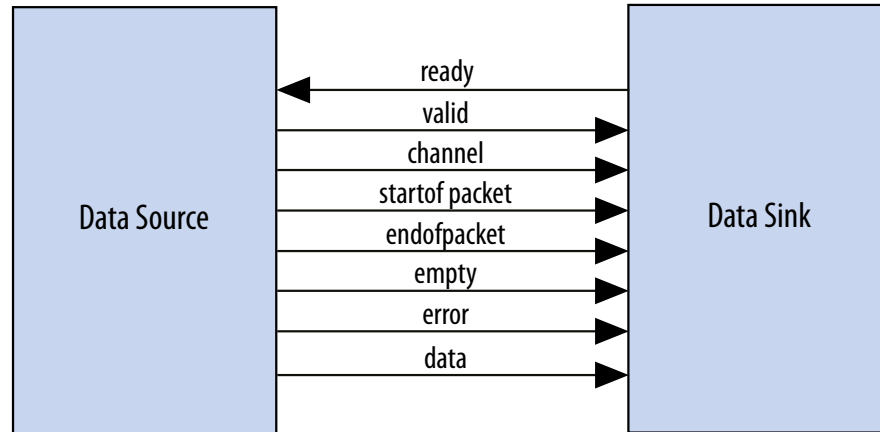
**Figure 230. Avalon-ST Connection Between the Source and Sink**

This source-sink pair includes only the data signal. The sink must be able to receive data as soon as the source interface comes out of reset.



**Figure 231. Signals Indicating the Start and End of Packets, Channel Numbers, Error Conditions, and Backpressure**

All data transfers using Avalon-ST interconnect occur synchronously on the rising edge of the associated clock interface. Throughput and frequency of a system depends on the components and how they are connected.



The IP Catalog includes Avalon-ST components that you can use to create datapaths, including datapaths whose input and output streams have different properties. Generated systems that include memory-mapped master and slave components may also use these Avalon-ST components because Platform Designer generation creates interconnect with a structure similar to a network topology, as described in *Platform Designer Transformations*. The following sections introduce the Avalon-ST components.

#### Related Links

[Platform Designer Transformations](#) on page 664

### 11.2.1 Avalon-ST Adapters

Platform Designer automatically adds Avalon-ST adapters between two components during system generation when it detects mismatched interfaces. If you connect mismatched Avalon-ST sources and sinks, for example, a 32-bit source and an 8-bit sink, Platform Designer inserts the appropriate adapter type to connect the mismatched interfaces.

After generation, you can view the inserted adapters selecting **System > Show System With Platform Designer Interconnect**. For each mismatched source-sink pair, Platform Designer inserts an Avalon-ST Adapter. The adapter instantiates the necessary adaptation logic as sub-components. You can review the logic for each adapter instantiation in the Hierarchy view by expanding each adapter's source and sink interface and comparing the relevant ports. For example, to determine why a channel adapter is inserted, expand the channel adapter's sink and source interfaces and review the channel port properties for each interface.

You can turn off the auto-inserted adapters feature by adding the `qsys_enable_avalon_streaming_transform=off` command to the `quartus.ini` file. When you turn off the auto-inserted adapters feature, if mismatched interfaces are detected during system generation, Platform Designer does not insert adapters and reports the mismatched interface with validation error message.



*Note:* The auto-inserted adapters feature does not work for video IP core connections.

### 11.2.1.1 Avalon-ST Adapter

The Avalon-ST adapter combines the logic of the channel, error, data format, and timing adapters. The Avalon-ST adapter provides adaptations between interfaces that have mismatched Avalon-ST endpoints. Based on the source and sink interface parameterizations for the Avalon-ST adapter, Platform Designer instantiates the necessary adapter logic (channel, error, data format, or timing) as hierarchical sub-components.

#### 11.2.1.1.1 Avalon-ST Adapter Parameters Common to Source and Sink Interfaces

**Table 134. Avalon-ST Adapter Parameters Common to Source and Sink Interfaces**

Parameter Name	Description
<b>Symbol Width</b>	Width of a single symbol in bits.
<b>Use Packet</b>	Indicates whether the source and sink interfaces connected to the adapter's source and sink interfaces include the <code>startofpacket</code> and <code>endofpacket</code> signals, and the optional <code>empty</code> signal.

#### 11.2.1.1.2 Avalon-ST Adapter Upstream Source Interface Parameters

**Table 135. Avalon-ST Adapter Upstream Source Interface Parameters**

Parameter Name	Description
<b>Source Data Width</b>	Controls the data width of the source interface <code>data</code> port.
<b>Source Top Channel</b>	Maximum number of output channels allowed.
<b>Source Channel Port Width</b>	Sets the bit width of the source interface <code>channel</code> port. If set to 0, there is no <code>channel</code> port on the sink interface.
<b>Source Error Port Width</b>	Sets the bit width of the source interface <code>error</code> port. If set to 0, there is no <code>error</code> port on the sink interface.
<b>Source Error Descriptors</b>	A list of strings that describe the error conditions for each bit of the source interface <code>error</code> signal.
<b>Source Uses Empty Port</b>	Indicates whether the source interface includes the <code>empty</code> port, and whether the sink interface should also include the <code>empty</code> port.
<b>Source Empty Port Width</b>	Indicates the bit width of the source interface <code>empty</code> port, and sets the bit width of the sink interface <code>empty</code> port.
<b>Source Uses Valid Port</b>	Indicates whether the source interface connected to the sink interface uses the <code>valid</code> port, and if set, configures the sink interface to use the <code>valid</code> port.
<b>Source Uses Ready Port</b>	Indicates whether the sink interface uses the <code>ready</code> port, and if set, configures the source interface to use the <code>ready</code> port.
<b>Source Ready Latency</b>	Specifies what ready latency to expect from the source interface connected to the adapter's sink interface.



### 11.2.1.1.3 Avalon-ST Adapter Downstream Sink Interface Parameters

**Table 136. Avalon-ST Adapter Downstream Sink Interface Parameters**

Parameter Name	Description
<b>Sink Data Width</b>	Indicates the bit width of the <code>data</code> port on the sink interface connected to the source interface.
<b>Sink Top Channel</b>	Maximum number of output channels allowed.
<b>Sink Channel Port Width</b>	Indicates the bit width of the <code>channel</code> port on the sink interface connected the source interface.
<b>Sink Error Port Width</b>	Indicates the bit width of the <code>error</code> port on the sink interface connected to the adapter's source interface. If set to zero, there is no error port on the source interface.
<b>Sink Error Descriptors</b>	A list of strings that describe the error conditions for each bit of the <code>error</code> port on the sink interface connected to the source interface.
<b>Sink Uses Empty Port</b>	Indicates whether the sink interface connected to the source interface uses the <code>empty</code> port, and whether the source interface should also use the <code>empty</code> port.
<b>Sink Empty Port Width</b>	Indicates the bit width of the <code>empty</code> port on the sink interface connected to the source interface, and configures a corresponding <code>empty</code> port on the source interface.
<b>Sink Uses Valid Port</b>	Indicates whether the sink interface connected to the source interface uses the <code>valid</code> port, and if set, configures the source interface to use the <code>valid</code> port.
<b>Sink Uses Ready Port</b>	Indicates whether the <code>ready</code> port on the sink interface is connected to the source interface, and if set, configures the sink interface to use the <code>ready</code> port.
<b>Sink Ready Latency</b>	Specifies what ready latency to expect from the source interface connected to the sink interface.

### 11.2.1.2 Channel Adapter

The channel adapter provides adaptations between interfaces that have different channel signal widths.

**Table 137. Channel Adapter Adaptations**

Condition	Description of Adapter Logic
The source uses channels, but the sink does not.	Platform Designer gives a warning at generation time. The adapter provides a simulation error and signals an error for data for any channel from the source other than 0.
The sink has channel, but the source does not.	Platform Designer gives a warning at generation time, and the channel inputs to the sink are all tied to a logical 0.
The source and sink both support channels, and the source's maximum channel number is less than the sink's maximum channel number.	The source's channel is connected to the sink's channel unchanged. If the sink's channel signal has more bits, the higher bits are tied to a logical 0.
The source and sink both support channels, but the source's maximum channel number is greater than the sink's maximum channel number.	The source's channel is connected to the sink's channel unchanged. If the source's channel signal has more bits, the higher bits are left unconnected. Platform Designer gives a warning that channel information may be lost. An adapter provides a simulation error message and an error indication if the value of channel from the source is greater than the sink's maximum number of channels. In addition, the <code>valid</code> signal to the sink is deasserted so that the sink never sees data for channels that are out of range.



### 11.2.1.2.1 Avalon-ST Channel Adapter Input Interface Parameters

**Table 138. Avalon-ST Channel Adapter Input Interface Parameters**

Parameter Name	Description
<b>Channel Signal Width (bits)</b>	Width of the input channel signal in bits
<b>Max Channel</b>	Maximum number of input channels allowed.

### 11.2.1.2.2 Avalon-ST Channel Adapter Output Interface Parameters

**Table 139. Avalon-ST Channel Adapter Output Interface Parameters**

Parameter Name	Description
<b>Channel Signal Width (bits)</b>	Width of the output channel signal in bits.
<b>Max Channel</b>	Maximum number of output channels allowed.

### 11.2.1.2.3 Avalon-ST Channel Adapter Common to Input and Output Interface Parameters

**Table 140. Avalon-ST Channel Adapter Common to Input and Output Interface Parameters**

Parameter Name	Description
<b>Data Bits Per Symbol</b>	Number of bits for each symbol in a transfer.
<b>Include Packet Support</b>	When the Avalon-ST Channel adapter supports packets, the <code>startofpacket</code> , <code>endofpacket</code> , and optional <code>empty</code> signals are included on its sink and source interfaces.
<b>Include Empty Signal</b>	Indicates whether an <code>empty</code> signal is required.
<b>Data Symbols Per Beat</b>	Number of symbols per transfer.
<b>Support Backpressure with the ready signal</b>	Indicates whether a <code>ready</code> signal is required.
<b>Ready Latency</b>	Specifies the ready latency to expect from the sink connected to the module's source interface.
<b>Error Signal Width (bits)</b>	Bit width of the <code>error</code> signal.
<b>Error Signal Description</b>	A list of strings that describes what each bit of the <code>error</code> signal represents.

### 11.2.1.3 Data Format Adapter

The data format adapter allows you to connect interfaces that have different values for the parameters defining the `data` signal, or interfaces where the source does not use the `empty` signal, but the sink does use the `empty` signal. One of the most common uses of this adapter is to convert data streams of different widths.

**Table 141. Data Format Adapter Adaptations**

Condition	Description of Adapter Logic
The source and sink's bits per symbol parameters are different.	The connection cannot be made.
The source and sink have a different number of symbols per beat.	The adapter converts the source's width to the sink's width.

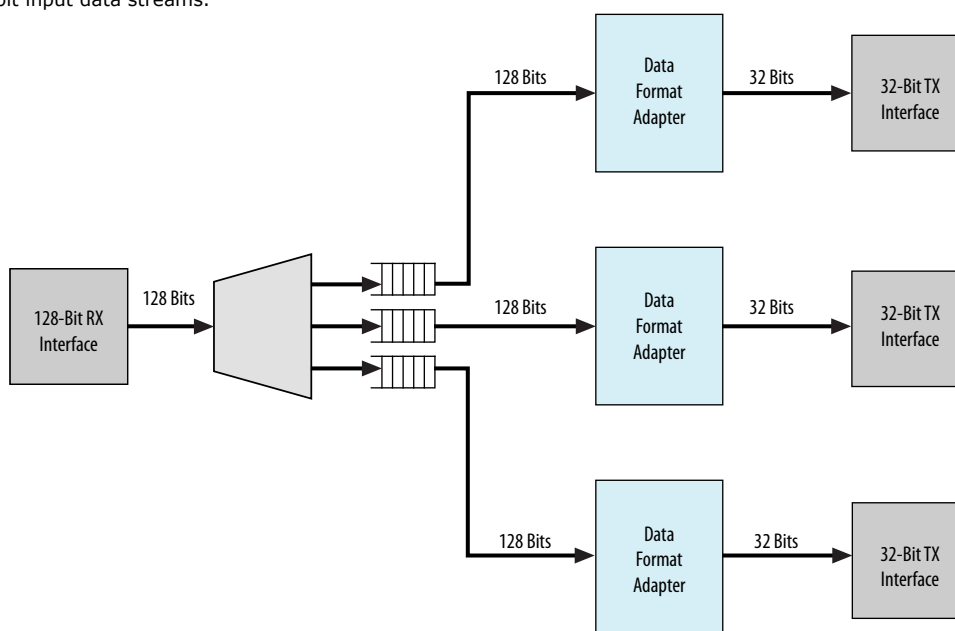
*continued...*



Condition	Description of Adapter Logic
	If the adaptation is from a wider to a narrower interface, a beat of data at the input corresponds to multiple beats of data at the output. If the input <code>error</code> signal is asserted for a single beat, it is asserted on output for multiple beats. If the adaptation is from a narrow to a wider interface, multiple input beats are required to fill a single output beat, and the output <code>error</code> is the logical OR of the input <code>error</code> signal.
The source uses the <code>empty</code> signal, but the sink does not use the <code>empty</code> signal.	Platform Designer cannot make the connection.

**Figure 232. Avalon Streaming Interconnect with Data Format Adapter**

In this example, the data format adapter allows a connection between a 128-bit output data stream and three 32-bit input data streams.



### 11.2.1.3.1 Avalon-ST Data Format Adapter Input Interface Parameters

**Table 142. Avalon-ST Data Format Adapter Input Interface Parameters**

Parameter Name	Description
<b>Data Symbols Per Beat</b>	Number of symbols per transfer.
<b>Include Empty Signal</b>	Indicates whether an <code>empty</code> signal is required.

### 11.2.1.3.2 Avalon-ST Data Format Adapter Output Interface Parameters

**Table 143. Avalon-ST Data Format Adapter Output Interface Parameters**

Parameter Name	Description
<b>Data Symbols Per Beat</b>	Number of symbols per transfer.
<b>Include Empty Signals</b>	Indicates whether an <code>empty</code> signal is required.

### 11.2.1.3.3 Avalon-ST Data Format Adapter Common to Input and Output Interface Parameters

**Table 144. Avalon-ST Data Format Adapter Common to Input and Output Interface Parameters**

Parameter Name	Description
<b>Data Bits Per Symbol</b>	Number of bits for each symbol in a transfer.
<b>Include Packet Support</b>	When the Avalon-ST Data Format adapter supports packets, Platform Designer uses <code>startofpacket</code> , <code>endofpacket</code> , and <code>empty</code> signals.
<b>Channel Signal Width (bits)</b>	Width of the output channel signal in bits.
<b>Max Channel</b>	Maximum number of channels allowed.
<b>Read Latency</b>	Specifies the ready latency to expect from the sink connected to the module's source interface.
<b>Error Signal Width (bits)</b>	Width of the <code>error</code> signal output in bits.
<b>Error Signal Description</b>	A list of strings that describes what each bit of the <code>error</code> signal represents.

### 11.2.1.4 Error Adapter

The error adapter ensures that per-bit-error information provided by the source interface is correctly connected to the sink interface's input error signal. Error conditions that both source and sink can process are connected. If the source has an `error` signal representing an error condition that is not supported by the sink, the signal is left unconnected; the adapter provides a simulation error message and an error indication if the error is asserted. If the sink has an error condition that is not supported by the source, the sink's input error bit corresponding to that condition is set to 0.

*Note:* The output interface error signal descriptor accepts an error set with an `other` descriptor. Platform Designer assigns the bit-wise ORing of all input error bits that are unmatched, to the output interface error bits set with the `other` descriptor.

#### 11.2.1.4.1 Avalon-ST Error Adapter Input Interface Parameters

**Table 145. Avalon-ST Error Adapter Input Interface Parameters**

Parameter Name	Description
<b>Error Signal Width (bits)</b>	The width of the <code>error</code> signal. Valid values are 0–256 bits. Type 0 if the <code>error</code> signal is not used.
<b>Error Signal Description</b>	The description for each of the error bits. If scripting, separate the description fields by commas. For a successful connection, the description strings of the error bits in the source and sink must match and are case sensitive.



### 11.2.1.4.2 Avalon-ST Error Adapter Output Interface Parameters

**Table 146. Avalon-ST Error Adapter Output Interface Parameters**

Parameter Name	Description
<b>Error Signal Width (bits)</b>	The width of the <code>error</code> signal. Valid values are 0–256 bits. Type 0 if you do not need to send error values.
<b>Error Signal Description</b>	The description for each of the error bits. Separate the description fields by commas. For successful connection, the description of the error bits in the source and sink must match, and are case sensitive.

### 11.2.1.4.3 Avalon-ST Error Adapter Common to Input and Output Interface Parameters

**Table 147. Avalon-ST Error Adapter Common to Input and Output Interface Parameters**

Parameter Name	Description
<b>Support Backpressure with the ready signal</b>	Turn on this option to add the backpressure functionality to the interface.
<b>Ready Latency</b>	When the <code>ready</code> signal is used, the value for <code>ready_latency</code> indicates the number of cycles between when the <code>ready</code> signal is asserted and when valid data is driven.
<b>Channel Signal Width (bits)</b>	The width of the <code>channel</code> signal. A channel width of 4 allows up to 16 channels. The maximum width of the <code>channel</code> signal is eight bits. Set to 0 if channels are not used.
<b>Max Channel</b>	The maximum number of channels that the interface supports. Valid values are 0–255.
<b>Data Bits Per Symbol</b>	Number of bits per symbol.
<b>Data Symbols Per Beat</b>	Number of symbols per active transfer.
<b>Include Packet Support</b>	Turn on this option if the connected interfaces support a packet protocol, including the <code>startofpacket</code> , <code>endofpacket</code> and <code>empty</code> signals.
<b>Include Empty Signal</b>	Turn this option on if the cycle that includes the <code>endofpacket</code> signal can include empty symbols. This signal is not necessary if the number of symbols per beat is 1.

### 11.2.1.5 Timing Adapter

The timing adapter allows you to connect component interfaces that require a different number of cycles before driving or receiving data. This adapter inserts a FIFO buffer between the source and sink to buffer data or pipeline stages to delay the back-pressure signals. You can also use the timing adapter to connect interfaces that support the `ready` signal, and those that do not. The timing adapter treats all signals other than the `ready` and `valid` signals as payload, and simply drives them from the source to the sink.

**Table 148. Timing Adapter Adaptations**

Condition	Adaptation
The source has <code>ready</code> , but the sink does not.	In this case, the source can respond to <code>backpressure</code> , but the sink never needs to apply it. The <code>ready</code> input to the source interface is connected directly to logical 1.
The source does not have <code>ready</code> , but the sink does.	The sink may apply <code>backpressure</code> , but the source is unable to respond to it. There is no logic that the adapter can insert that prevents data loss when the source asserts <code>valid</code> but the sink is not ready. The adapter provides simulation time error messages if data is lost. The user is presented with a warning, and the connection is allowed.
The source and sink both support <code>backpressure</code> , but the sink's ready latency is greater than the source's.	The source responds to <code>ready</code> assertion or deassertion faster than the sink requires it. The number of pipeline stages equal to the difference in ready latency are inserted in the <code>ready</code> path from the sink back to the source, causing the source and the sink to see the same cycles as <code>ready</code> cycles.
The source and sink both support <code>backpressure</code> , but the sink's ready latency is less than the source's.	The source cannot respond to <code>ready</code> assertion or deassertion in time to satisfy the sink. A FIFO whose depth is equal to the difference in ready latency is inserted to compensate for the source's inability to respond in time.

### 11.2.1.5.1 Avalon-ST Timing Adapter Input Interface Parameters

**Table 149. Avalon-ST Timing Adapter Input Interface Parameters**

Parameter Name	Description
<b>Support Backpressure with the ready signal</b>	Indicates whether a <code>ready</code> signal is required.
<b>Read Latency</b>	Specifies the ready latency to expect from the sink connected to the module's source interface.
<b>Include Valid Signal</b>	Indicates whether the sink interface requires a valid signal.

### 11.2.1.5.2 Avalon-ST Timing Adapter Output Interface Parameters

**Table 150. Avalon-ST Timing Adapter Output Interface Parameters**

Parameter Name	Description
<b>Support Backpressure with the ready signal</b>	Indicates whether a <code>ready</code> signal is required.
<b>Read Latency</b>	Specifies the ready latency to expect from the sink connected to the module's source interface.
<b>Include Valid Signal</b>	Indicates whether the sink interface requires a valid signal.

### 11.2.1.5.3 Avalon-ST Timing Adapter Common to Input and Output Interface Parameters

**Table 151. Avalon-ST Timing Adapter Common to Input and Output Interface Parameters**

Parameter Name	Description
<b>Data Bits Per Symbol</b>	Number of bits for each symbol in a transfer.
<b>Include Packet Support</b>	Turn this option on if the connected interfaces support a packet protocol, including the <code>startofpacket</code> , <code>endofpacket</code> and <code>empty</code> signals.
<b>Include Empty Signal</b>	Turn this option on if the cycle that includes the <code>endofpacket</code> signal can include empty symbols. This signal is not necessary if the number of symbols per beat is 1.
<b>Data Symbols Per Beat</b>	Number of symbols per active transfer.
<i>continued...</i>	



Parameter Name	Description
Channel Signal Width (bits)	Width of the output channel signal in bits.
Max Channel	Maximum number of output channels allowed.
Error Signal Width (bits)	Width of the output <code>error</code> signal in bits.
Error Signal Description	A list of strings that describes errors.

## 11.3 Interrupt Interfaces

Using individual requests, the interrupt logic can process up to 32 IRQ inputs connected to each interrupt receiver. With this logic, the interrupt sender connected to interrupt `receiver_0` is the highest priority with sequential receivers being successively lower priority. You can redefine the priority of interrupt senders by instantiating the IRQ mapper component. For more information refer to *IRQ Mapper*.

You can define the interrupt sender interface as asynchronous with no associated clock or reset interfaces. You can also define the interrupt receiver interface as asynchronous with no associated clock or reset interfaces. As a result, the receiver does its own synchronization internally. Platform Designer does not insert interrupt synchronizers for such receivers.

For clock crossing adaption on interrupts, Platform Designer inserts a synchronizer, which is clocked with the interrupt end point interface clock when the corresponding starting point interrupt interface has no clock or a different clock (than the end point). Platform Designer inserts the adapter if there is any kind of mismatch between the start and end points. Platform Designer does not insert the adapter if the interrupt receiver does not have an associated clock.

### Related Links

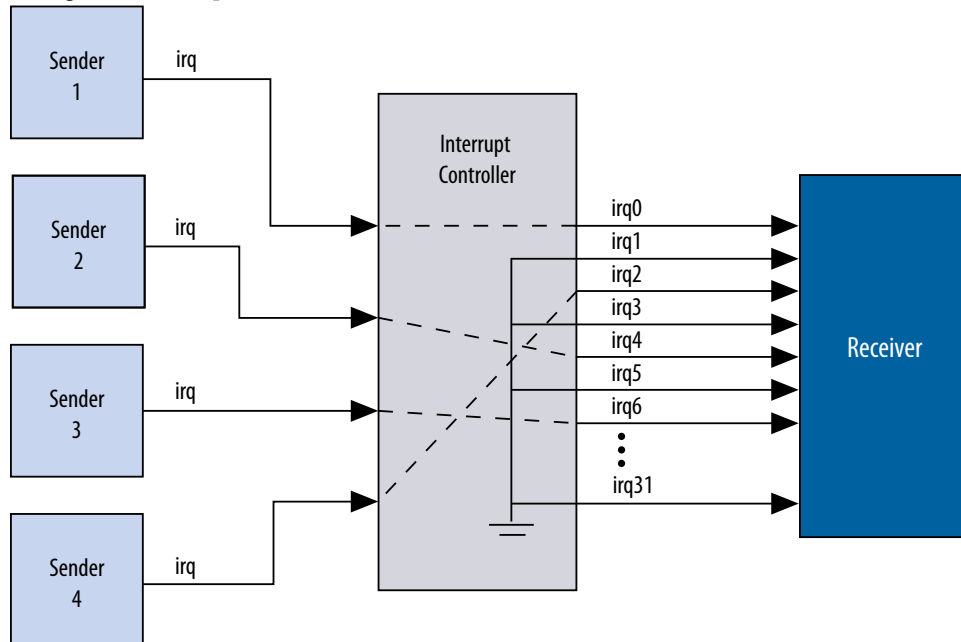
[IRQ Mapper](#) on page 695

### 11.3.1 Individual Requests IRQ Scheme

In the individual requests IRQ scheme, Platform Designer interconnect passes IRQs directly from the sender to the receiver, without making assumptions about IRQ priority. If multiple senders assert their IRQs simultaneously, the receiver logic determines which IRQ has highest priority, and then responds appropriately.

**Figure 233. Interrupt Controller Mapping IRQs**

Using individual requests, the interrupt controller can process up to 32 IRQ inputs. The interrupt controller generates a 32-bit signal `irq[31:0]` to the receiver, and maps slave IRQ signals to the bits of `irq[31:0]`. Any unassigned bits of `irq[31:0]` are disabled.



### 11.3.2 Assigning IRQs in Platform Designer

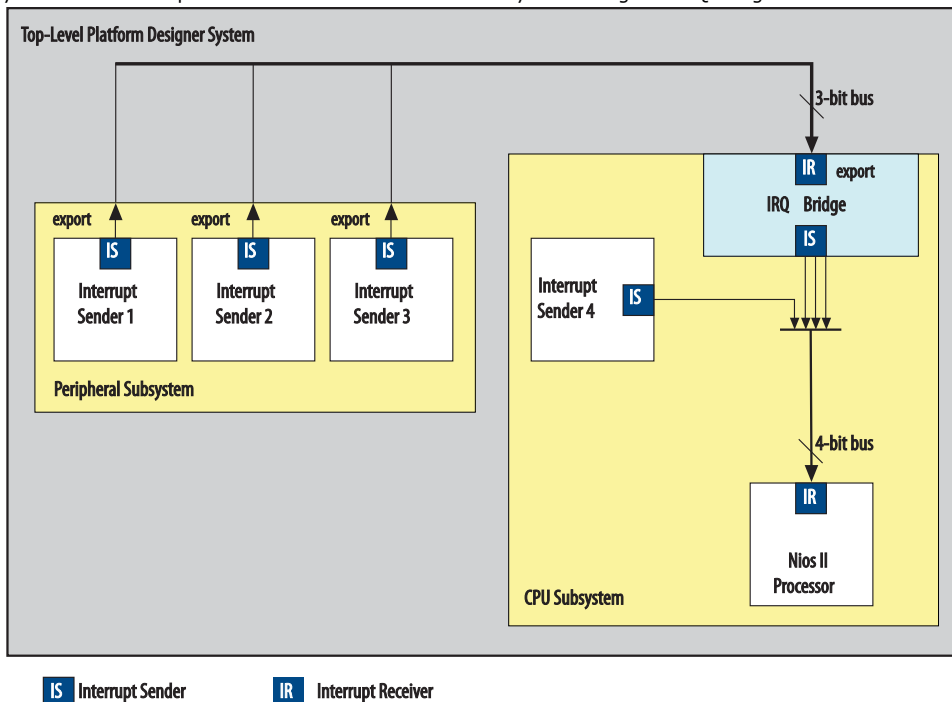
You assign IRQ connections on the **System Contents** tab of Platform Designer. After adding all components to the system, you connect interrupt senders and receivers. You can use the **IRQ** column to specify an IRQ number with respect to each receiver, or to specify a receiver's IRQ as unconnected. Platform Designer uses the following three components to implement interrupt handling: IRQ Bridge, IRQ Mapper, and IRQ Clock Crosser.

#### 11.3.2.1 IRQ Bridge

The IRQ Bridge allows you to route interrupt wires between Platform Designer subsystems.

**Figure 234. Platform Designer IRQ Bridge Application**

The peripheral subsystem example below has three interrupt senders that are exported to the to- level of the subsystem. The interrupts are then routed to the CPU subsystem using the IRQ bridge.



*Note:*

Nios II BSP tools support the IRQ Bridge. Interrupts connected via an IRQ Bridge appear in the generated `system.h` file. You can use the following properties with the IRQ Bridge, which do not effect Platform Designer interconnect generation. Platform Designer uses these properties to generate the correct IRQ information for downstream tools:

- `set_interface_property <sender port> bridgesToReceiver <receiver port>`— The `<sender port>` of the IP generates a signal that is received on the IP's `<receiver port>`. Sender ports are single bits. Receivers ports can be multiple bits. Platform Designer requires the `bridgedReceiverOffset` property to identify the `<receiver port>` bit that the `<sender port>` sends.
- `set_interface_property <sender port> bridgedReceiverOffset <port number>`— Indicates the `<port number>` of the receiver port that the `<sender port>` sends.

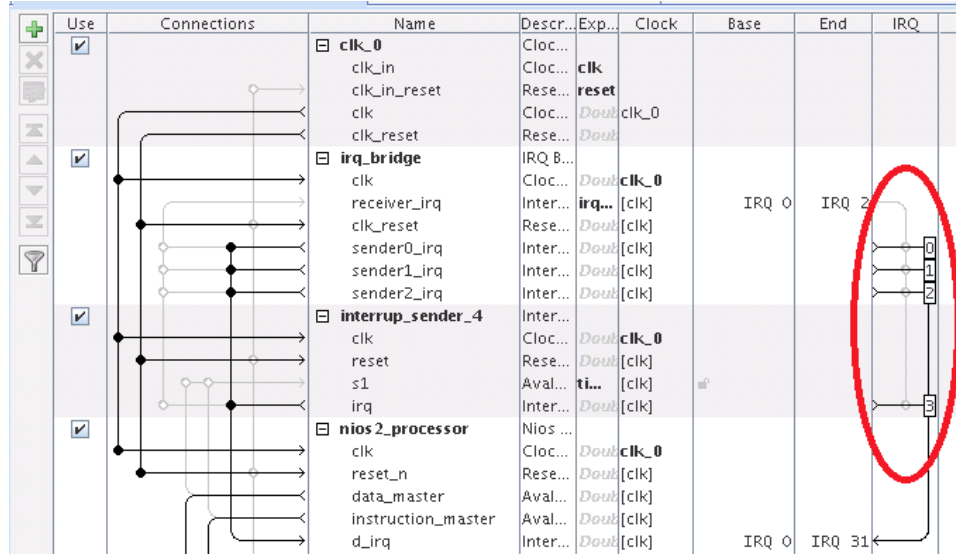
### 11.3.2.2 IRQ Mapper

Platform Designer inserts the IRQ Mapper automatically during generation. The IRQ Mapper converts individual interrupt wires to a bus, and then maps the appropriate IRQ priority number onto the bus.

By default, the interrupt sender connected to the `receiver0` interface of the IRQ mapper is the highest priority, and sequential receivers are successively lower priority. You can modify the interrupt priority of each IRQ wire by modifying the IRQ priority number in Platform Designer under the **IRQ** column. The modified priority is reflected in the **IRQ\_MAP** parameter for the auto-inserted IRQ Mapper.

**Figure 235. IRQ Column in Platform Designer**

Circled in the **IRQ** column are the default interrupt priorities allocated for the CPU subsystem.



**Related Links**

[IRQ Bridge](#) on page 694

**11.3.2.3 IRQ Clock Crosser**

The IRQ Clock Crosser synchronizes interrupt senders and receivers that are in different clock domains. To use this component, connect the clocks for both the interrupt sender and receiver, and for both the interrupt sender and receiver interfaces. Platform Designer automatically inserts this component when it is required.

**11.4 Clock Interfaces**

Clock interfaces define the clocks used by a component. Components can have clock inputs, clock outputs, or both. To update the clock frequency of the component, use the **Parameters** tab for the clock source.





The **Clock Source** parameters allows you to set the following options:

- **Clock frequency**—The frequency of the output clock from this clock source.
- **Clock frequency is known**— When turned on, the clock frequency is known. When turned off, the frequency is set from outside the system.  
*Note:* If turned off, system generation may fail because the components do not receive the necessary clock information. For best results, turn this option on before system generation.
- **Reset synchronous edges**
  - **None**—The reset is asserted and deasserted asynchronously. You can use this setting if you have internal synchronization circuitry that matches the reset required for the IP in the system.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously and asserted asynchronously.

For more information about synchronous design practices, refer to *Recommended Design Practices*

#### Related Links

[Recommended Design Practices](#) on page 152

### 11.4.1 (High Speed Serial Interface) HSSI Clock Interfaces

You can use HSSI Serial Clock and HSSI Bonded Clock interfaces in Platform Designer to enable high speed serial connectivity between clocks that are used by certain IP protocols.

#### 11.4.1.1 HSSI Serial Clock Interface

You can connect the HSSI Serial Clock interface with only similar type of interfaces, for example, you can connect a HSSI Serial Clock Source interface to a HSSI Serial Clock Sink interface.

##### 11.4.1.1.1 HSSI Serial Clock Source

The HSSI Serial Clock interface includes a source in the **Start** direction.

You can instantiate the HSSI Serial Clock Source interface in the `_hw.tcl` file as:

```
add_interface <name> hssi_serial_clock start
```

You can connect the HSSI Serial Clock Source to multiple HSSI Serial Clock Sinks because the HSSI Serial Clock Source supports multiple fan-outs. This Interface has a single **clk** port role limited to a 1 bit width, and a **clockRate** parameter, which is the frequency of the clock driven by the HSSI Serial Clock Source interface.

An unconnected and unexported HSSI Serial Source is valid and does not generate error messages.

**Table 152. HSSI Serial Clock Source Port Roles**

Name	Direction	Width	Description
clk	Output	1 bit	A single bit wide port role, which provides synchronization for internal logic.

**Table 153. HSSI Serial Clock Source Parameters**

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven byte HSSI Serial Clock Source interface.

#### 11.4.1.1.2 HSSI Serial Clock Sink

The HSSI Serial Clock interface includes a sink in the **End** direction.

You can instantiate the HSSI Serial Clock Sink interface in the `_hw.tcl` file as:

```
add_interface <name> hssi_serial_clock end
```

You can connect the HSSI Serial Clock Sink interface to a single HSSI Serial Clock Source interface; you cannot connect it to multiple sources. This Interface has a single **clk** port role limited to a 1 bit width, and a **clockRate** parameter, which is the frequency of the clock driven by the HSSI Serial Clock Source interface.

An unconnected and unexported HSSI Serial Sink is invalid and generates error messages.

**Table 154. HSSI Serial Clock Sink Port Roles**

Name	Direction	Width	Description
clk	Output	1	A single bit wide port role, which provides synchronization for internal logic

**Table 155. HSSI Serial Clock Sink Parameters**

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven by the HSSI Serial Clock Source interface. When you specify a <b>clockRate</b> greater than 0, then this interface can be driven only at that rate.

#### 11.4.1.1.3 HSSI Serial Clock Connection

The HSSI Serial Clock Connection defines a connection between a HSSI Serial Clock Source connection point, and a HSSI Serial Clock Sink connection point.

A valid HSSI Serial Clock Connection exists when all the following criteria are satisfied. If the following criteria are not satisfied, Platform Designer generates error messages and the connection is prohibited.

- The starting connection point is an HSSI Serial Clock Source with a single port role **clk** and maximum 1 bit in width. The direction of the starting port is **Output**.
- The ending connection point is an HSSI Serial Clock Sink with a single port role **clk**, and maximum 1 bit in width. The direction of the ending port is **Input**.
- If the parameter, **clockRate** of the HSSI Serial Clock Sink is greater than 0, the connection is only valid if the **clockRate** of the HSSI Serial Clock Source is the same as the **clockRate** of the HSSI Serial Clock Sink.



#### 11.4.1.1.4 HSSI Serial Clock Example

##### Example 99. HSSI Serial Clock Interface Example

You can make connections to declare the HSSI Serial Clock interfaces in the `_hw.tcl`.

```
package require -exact qsys 14.0

set_module_property name hssi_serial_component
set_module_property ELABORATION_CALLBACK elaborate

add_fileset QUARTUS_SYNTH QUARTUS_SYNTH generate
add_fileset SIM_VERILOG SIM_VERILOG generate
add_fileset SIM_VHDL SIM_VHDL generate

set_fileset_property QUARTUS_SYNTH TOP_LEVEL \
"hssi_serial_component"

set_fileset_property SIM_VERILOG TOP_LEVEL "hssi_serial_component"
set_fileset_property SIM_VHDL TOP_LEVEL "hssi_serial_component"

proc elaborate {} {
    # declaring HSSI Serial Clock Source
    add_interface my_clock_start hssi_serial_clock start
    set_interface_property my_clock_start ENABLED true

    add_interface_port my_clock_start hssi_serial_clock_port_out \
    clk Output 1

    # declaring HSSI Serial Clock Sink
    add_interface my_clock_end hssi_serial_clock end
    set_interface_property my_clock_end ENABLED true

    add_interface_port my_clock_end hssi_serial_clock_port_in clk \
    Input 1
}

proc generate { output_name } {
    add_fileset_file hssi_serial_component.v VERILOG PATH \
    "hssi_serial_component.v"
}
```

##### Example 100. HSSI Serial Clock Instantiated in a Composed Component

If you use the components in a hierarchy, for example, instantiated in a composed component, you can declare the connections as illustrated in this example.

```
add_instance myinst1 hssi_serial_component
add_instance myinst2 hssi_serial_component
# add connection from source of myinst1 to sink of myinst2

add_connection myinst1.my_clock_start myinst2.my_clock_end \
hssi_serial_clock

# adding connection from source of myinst2 to sink of myinst1

add_connection myinst2.my_clock_start myinst2.my_clock_end \
hssi_serial_clock
```

#### 11.4.1.2 HSSI Bonded Clock Interface

You can connect the HSSI Bonded Clock interface only with similar type of interfaces, for example, you can connect a HSSI Bonded Clock Source interface to a HSSI Bonded Clock Sink interface.

### 11.4.1.2.1 HSSI Bonded Clock Source

The HSSI Bonded Clock interface includes a source in the **Start** direction.

You can instantiate the HSSI Bonded Clock Source interface in the `_hw.tcl` file as:

```
add_interface <name> hssi_bonded_clock start
```

You can connect the HSSI Bonded Clock Source to multiple HSSI Bonded Clock Sinks because the HSSI Serial Clock Source supports multiple fanouts. This Interface has a single **clk** port role limited to a width range of 1 to 1024 bits. The HSSI Bonded Clock Source interface has two parameters: **clockRate** and **serializationFactor**.

**clockRate** is the frequency of the clock driven by the HSSI Bonded Clock Source interface, and the **serializationFactor** is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the required frequency and phases of the individual clocks within the HSSI Bonded Clock interface

An unconnected and unexported HSSI Bonded Source is valid, and does not generate error messages.

**Table 156. HSSI Bonded Clock Source Port Roles**

Name	Direction	Width	Description
clk	Output	1 to 24 bits	A multiple bit wide port role which provides synchronization for internal logic.

**Table 157. HSSI Bonded Clock Source Parameters**

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven by HSSI Serial Clock Source interface.
serialization	long	0	No	The serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the necessary frequency and phases of the individual clocks within the HSSI Bonded Clock interface.

### 11.4.1.2.2 HSSI Bonded Clock Sink

The HSSI Bonded Clock interface includes a sink in the **End** direction.

You can instantiate the HSSI Bonded Clock Sink interface in the `_hw.tcl` file as:

```
add_interface <name> hssi_bonded_clock end
```

You can connect the HSSI Bonded Clock Sink interface to a single HSSI Bonded Clock Source interface; you cannot connect it to multiple sources. This Interface has a single **clk** port role limited to a width range of 1 to 1024 bits. The HSSI Bonded Clock Source interface has two parameters: **clockRate** and **serializationFactor**. **clockRate** is the frequency of the clock driven by the HSSI Bonded Clock Source interface, and the serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the required frequency and phases of the individual clocks within the HSSI Bonded Clock interface

An unconnected and unexported HSSI Bonded Sink is invalid and generates error messages.



**Table 158. HSSI Bonded Clock Source Port Roles**

Name	Direction	Width	Description
clk	Output	1 to 24 bits	A multiple bit wide port role which provides synchronization for internal logic.

**Table 159. HSSI Bonded Clock Source Parameters**

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven by HSSI Serial Clock Source interface.
serialization	long	0	No	The serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the necessary frequency and phases of the individual clocks within the HSSI Bonded Clock interface.

#### 11.4.1.2.3 HSSI Bonded Clock Connection

The HSSI Bonded Clock Connection defines a connection between a HSSI Bonded Clock Source connection point, and a HSSI Bonded Clock Sink connection point.

A valid HSSI Bonded Clock Connection exists when all the following criteria are satisfied. If the following criteria are not satisfied, Platform Designer generates error messages and the connection is prohibited.

- The starting connection point is an HSSI Bonded Clock Source with a single port role **clk** with a width range of 1 to 24 bits. The direction of the starting port is **Output**.
- The ending connection point is an HSSI Bonded Clock Sink with a single port role **clk** with a width range of 1 to 24 bits. The direction of the ending port is **Input**.
- The width of the starting connection point **clk** must be the same as the width of the ending connection point.
- If the parameter, **clockRate** of the HSSI Bonded Clock Sink greater than 0, then the connection is only valid if the **clockRate** of the HSSI Bonded Clock Source is same as the **clockRate** of the HSSI Bonded Clock Sink.
- If the parameter, **serializationFactor** of the HSSI Bonded Clock Sink is greater than 0, Platform Designer generates a warning if the **serializationFactor** of HSSI Bonded Clock Source is not same as the **serializationFactor** of the HSSI Bonded Clock Sink.

#### 11.4.1.2.4 HSSI Bonded Clock Example

##### Example 101. HSSI Bonded Clock Interface Example

You can make connections to declare the HSSI Bonded Clock interfaces in the **\_hw.tcl** file.

```
package require -exact qsys 14.0

set_module_property name hssi_bonded_component
set_module_property ELABORATION_CALLBACK elaborate

add_fileset synthesis QUARTUS_SYNTH generate
add_fileset verilog_simulation SIM_VERILOG generate

set_fileset_property synthesis TOP_LEVEL "hssi_bonded_component"
```

```

set_fileset_property verilog_simulation TOP_LEVEL \
"hssi_bonded_component"

proc elaborate {} {
    add_interface my_clock_start hssi_bonded_clock start
    set_interface_property my_clock_start ENABLED true

    add_interface_port my_clock_start hssi_bonded_clock_port_out \
    clk Output 1024

    add_interface my_clock_end hssi_bonded_clock end
    set_interface_property my_clock_end ENABLED true

    add_interface_port my_clock_end hssi_bonded_clock_port_in \
    clk Input 1024
}

proc generate { output_name } {
    add_fileset_file hssi_bonded_component.v VERILOG PATH \
    "hssi_bonded_component.v"}

```

If you use the components in a hierarchy, for example, instantiated in a composed component, you can declare the connections as illustrated in this example.

### Example 102. HSII Bonded Clock Instantiated in a Composed Component

```

add_instance myinst1 hssi_bonded_component
add_instance myinst2 hssi_bonded_component
# add connection from source of myinst1 to sink of myinst2

add_connection myinst1.my_clock_start myinst2.my_clock_end \
hssi_bonded_clock

# adding connection from source of myinst2 to sink of myinst1

add_connection myinst2.my_clock_start myinst2.my_clock_end \
hssi_bonded_clock

```

## 11.5 Reset Interfaces

Reset interfaces provide both soft and hard reset functionality. Soft reset logic typically re-initializes registers and memories without powering down the device. Hard reset logic initializes the device after power-on. You can define separate reset sources for each clock domain, a single reset source for all clocks, or any combination in between.

You can choose to create a single global reset domain by selecting **Create Global Reset Network** on the System menu. If your design requires more than one reset domain, you can implement your own reset logic and connectivity. The IP Catalog includes a reset controller, reset sequencer, and a reset bridge to implement the reset functionality. You can also design your own reset logic.

**Note:** If you design your own reset circuitry, you must carefully consider situations which may result in system lockup. For example, if an Avalon-MM slave is reset in the middle of a transaction, the Avalon-MM master may lockup.



### 11.5.1 Single Global Reset Signal Implemented by Platform Designer

When you select **System > Create Global Reset Network**, the Platform Designer interconnect creates a global reset bus. All the reset requests are ORed together, synchronized to each clock domain, and fed to the reset inputs. The duration of the reset signal is at least one clock period.

The Platform Designer interconnect inserts the system-wide reset under the following conditions:

- The global reset input to the Platform Designer system is asserted.
- Any component asserts its `resetrequest` signal.

### 11.5.2 Reset Controller

Platform Designer automatically inserts a reset controller block if the input reset source does not have a reset request, but the connected reset sink requires a reset request.

The Reset Controller has the following parameters that you can specify to customize its behavior:

- **Number of inputs**— Indicates the number of individual reset interfaces the controller ORs to create a signal reset output.
- **Output reset synchronous edges**—Specifies the level of synchronization. You can select one of the following options:
  - **None**—The reset is asserted and deasserted asynchronously. You can use this setting if you have designed internal synchronization circuitry that matches the reset style required for the IP in the system.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously and asserted asynchronously.
- **Synchronization depth**—Specifies the number of register stages the synchronizer uses to eliminate the propagation of metastable events.
- **Reset request**—Enables reset request generation, which is an early signal that is asserted before reset assertion. The reset request is used by blocks that require protection from asynchronous inputs, for example, M20K blocks.

Platform Designer automatically inserts reset synchronizers under the following conditions:

- More than one reset source is connected to a reset sink
- There is a mismatch between the reset source's synchronous edges and the reset sinks' synchronous edges

### 11.5.3 Reset Bridge

The Reset Bridge allows you to use a reset signal in two or more subsystems of your Platform Designer system. You can connect one reset source to local components, and export one or more to other subsystems, as required.



The Reset Bridge parameters are used to describe the incoming reset and include the following options:

- **Active low reset**—When turned on, reset is asserted low.
- **Synchronous edges**—Specifies the level of synchronization and includes the following options:
  - **None**—The reset is asserted and deasserted asynchronously. Use this setting if you have internal synchronization circuitry.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously, and asserted asynchronously.
- **Number of reset outputs**—The number of reset interfaces that are exported.

*Note:*

Platform Designer supports multiple reset sink connections to a single reset source interface. However, there are situations in composed systems where an internally generated reset must be exported from the composed system in addition to being used to connect internal components. In this situation, you must declare one reset output interface as an export, and use another reset output to connect internal components.

#### 11.5.4 Reset Sequencer

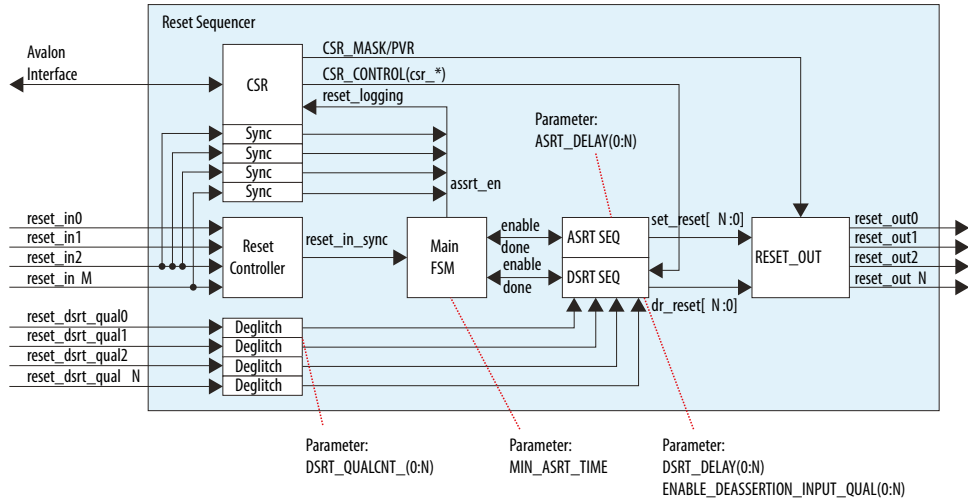
The Reset Sequencer allows you to control the assertion and deassertion sequence for Platform Designer system resets.

The Parameter Editor displays the expected assertion and deassertion sequences based on the current settings. You can connect multiple reset sources to the reset sequencer, and then connect the outputs of the Reset Sequencer to components in the system.





Figure 236. Elements and Flow of a Reset Sequencer



- Reset Controller—Reused reset controller block. It synchronizes the reset inputs into one and feeds into the main FSM of the sequencer block.
- Sync—Synchronization block (double flipflop).
- Deglitch—Deglitch block. This block waits for a signal to be at a level for X clocks before propagating the input to the output.
- CSR—This block contains the CSR Avalon interface and related CSR register and control block in the sequencer.
- Main FSM—Main sequencer. This block determines when assertion/deassertion and assertion hold timing occurs.
- [A/D]SRT SEQ—Generic sequencer block that sequences out assertion/deassertion of reset from 0:N. The block has multiple counters that saturate upon reaching count.
- RESET\_OUT—Controls the end output via:
  - Set/clear from the ASRT\_SEQ/DSRT\_SEQ.
  - Masking/forcing from CSR controls.
  - Remap of numbering (parameterization).

### 11.5.4.1 Reset Sequencer Parameters

Table 160. Reset Sequencer Parameters

Parameter	Description
<b>Number of reset outputs</b>	Sets the number of output resets to be sequenced, which is the number of output reset signals defined in the component with a range of 2 to 10.
<b>Number of reset inputs</b>	Sets the number of input reset signals to be sequenced, which is the number of input reset signals defined in the component with a range of 1 to 10.
<b>Minimum reset assertion time</b>	Specifies the minimum assertion cycles between the assertion of the last sequenced reset, and the deassertion of the first sequenced reset. The range is 0 to 1023.
<b>Enable Reset Sequencer CSR</b>	Enables CSR functionality of the Reset Sequencer through an Avalon interface.
<b>reset_out#</b>	Lists the reset output signals. Set the parameters in the other columns for each reset signal in the table.
<b>ASRT Seq#</b>	Determines the order of reset assertion. Enter the values 1, 2, 3, etc. to specify the required non-overlapping assertion order. This value determines the ASRT_REMAP value in the component HDL.
<b>ASRT Cycle#</b>	Number of cycles to wait before assertion of the reset. The value set here corresponds to the ASRT_DELAY value in the component HDL. The range is 0 to 1023.

*continued...*

Parameter	Description
<b>DSRT Seq#</b>	Determines the reset order of reset deassertion. Enter the values 1, 2, 3, etc. to specify the required non-overlapping deassertion order. This value determines the DSRT_REMAP value in the component HDL.
<b>DSRT Cycle#/Deglitch#</b>	Number of cycles to wait before deasserting or deglitching the reset. If the <b>USE_DSRT_QUAL</b> parameter is set to 0, specifies the number of cycles to wait before deasserting the reset. If <b>USE_DSRT_QUAL</b> is set to 1, specifies the number of cycles to deglitch the input <code>reset_dsrt_qual</code> signal. This value determines either the DSRT_DELAY, or the DSRT_QUALCNT value in the component HDL, depending on the <b>USE_DSRT_QUAL</b> parameter setting. The range is 0 to 1023.
<b>USE_DSRT_QUAL</b>	If you set <b>USE_DSRT_QUAL</b> to 1, the deassertion sequence waits for an external input signal for sequence qualification instead of waiting for a fixed delay count. To use a fixed delay count for deassertion, set this parameter to 0.

### 11.5.4.2 Reset Sequencer Timing Diagrams

Figure 237. Basic Sequencing

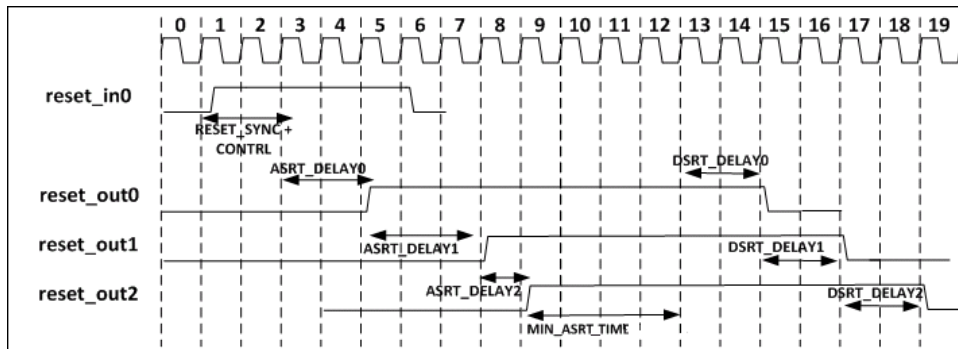
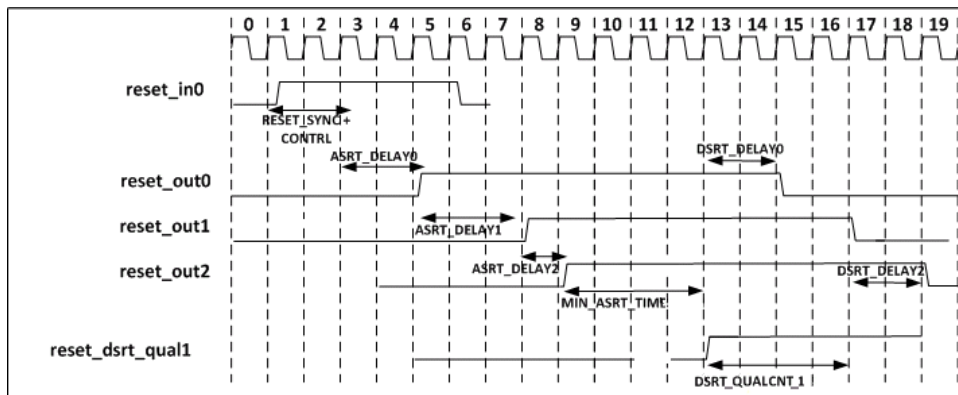


Figure 238. Sequencing with USE\_DSRT\_QUAL Set





### 11.5.4.3 Reset Sequencer CSR Registers

The Reset Sequencer's CSR registers provide the following functionality:

- **Support reset logging**
  - Ability to identify which reset is asserted.
  - Ability to determine whether any reset is currently active.
- **Support software triggered resets**
  - Ability to generate reset by writing to the register.
  - Ability to disable assertion or deassertion sequence.
- **Support software sequenced reset**
  - Ability for the software to fully control the assertion/deassertion sequence by writing to registers and stepping through the sequence.
- **Support reset override**
  - Ability to assert a specific component reset through software.

**Table 161. Reset Sequencer CSR Register Map**

Register	Offset	Width	Reset Value	Description
Status Register	0x00	32	0x0	The Status register indicates which sources are allowed to cause a reset.
Interrupt Enable Register	0x04	32	0x0	The Interrupt Enable register bits enable events triggering the IRQ of the reset sequencer.
Control Register	0x08	32	0x0	The Control register allows you to control the Reset Sequencer.
Software Sequenced Reset Assert Control Register	0x0C	32	0x3FF	You can program the Software Sequenced Reset Assert control register to control the reset assertion sequence.
Software Sequenced Reset Deassert Control Register	0x10	32	0x3FF	You can program the Software Sequenced Reset Deassert register to control the reset deassertion sequence.
Software Direct Controlled Resets	0x14	32	0x0	You can write a bit to 1 to assert the reset_outN signal, and to 0 to deassert the reset_outN signal.
Software Reset Masking	0x18	32	0x0	Masking off (writing 1) to a reset_outN "Reset Mask Enable" signal prevents the corresponding reset from being asserted. Writing a bit to 0 to a reset mask enable signal allows assertion of reset_outN.

#### 11.5.4.3.1 Reset Sequencer Status Register

The Status register indicates which sources are allowed to cause a reset.

You can clear bits by writing 1 to the bit location. The Reset Sequencer ignores attempts to write bits with a value of 0. If the sequencer is reset (power-on-reset), all bits are cleared, except the power-on-reset bit.

**Table 162. Values for the Status Register at Offset 0x00**

Bit	Attribute	Default	Description
31	RO	0	Reset Active—Indicates that the sequencer is currently active in reset sequence (assertion or deassertion).
30	RW1C	0	Reset Asserted and waiting for SW to proceed—Set when there is an active reset assertion, and the next sequence is waiting for the software to proceed. Only valid when the Enable SW sequenced reset assert option is turned on.
29	RW1C	0	Reset Deasserted and waiting for SW to proceed—Set when there is an active reset deassertion, and the next sequence is waiting for the software to proceed. Only valid when the Enable SW sequenced reset deassert option is turned on.
28:26	Reserved.		
25:16	RW1C	0	Reset deassertion input qualification signal <code>reset_dsrt_qual [9:0]</code> status—Indicates that the reset deassertion's input signal qualification signal is set. This bit is set on the detection of assertion of the signal.
15:12	Reserved.		
11	RW1C	0	<code>reset_in9</code> was triggered—Indicates that <code>reset_in9</code> triggered the reset. Software clears this bits by writing 1 to this location.
10	RW1C	0	<code>reset_in8</code> was triggered—Indicates that <code>reset_in8</code> triggered the reset. Software clears this bit by writing 1 to this location.
9	RW1C	0	<code>reset_in7</code> was triggered—Indicates that <code>reset_in7</code> triggered the reset. Software clears this bit by writing 1 to this location.
8	RW1C	0	<code>reset_in6</code> was triggered—Indicates that <code>reset_in6</code> triggered the reset. Software clears this bit by writing 1 to this location.
7	RW1C	0	<code>reset_in5</code> was triggered—Indicates that <code>reset_in5</code> triggered the reset. Software clears this bit by writing 1 to this location.
6	RW1C	0	<code>reset_in4</code> was triggered—Indicates that <code>reset_in4</code> triggered the reset. Software clears this bit by writing 1 to this location.
5	RW1C	0	<code>reset_in3</code> was triggered—Indicates that <code>reset_in3</code> triggered the reset. Software clears this bit by writing 1 to this location.
4	RW1C	0	<code>reset_in2</code> was triggered—Indicates that <code>reset_in2</code> triggered the reset. Software clears this bit by writing 1 to this location.
3	RW1C	0	<code>reset_in1</code> was triggered—Indicates that <code>reset_in1</code> triggered the reset. Software clears this bit by writing 1 to this location.
2	RW1C	0	<code>reset_in0</code> was triggered—Indicates that <code>reset_in0</code> triggered. Software clears this bit by writing 1 to this location.
1	RW1C	0	Software-triggered reset—Indicates that the software-triggered reset is set by the software, and triggering a reset.
0	RW1C	0	Power-on-reset was triggered—Asserted whenever the reset to the sequencer is triggered. This bit is NOT reset when sequencer is reset. Software clears this bit by writing 1 to this location.

### Related Links

[Reset Sequencer CSR Registers on page 707](#)



### 11.5.4.3.2 Reset Sequencer Interrupt Enable Register

The Interrupt Enable register bits enable events triggering the IRQ of the reset sequencer.

**Table 163. Values for the Interrupt Enable Register at Offset 0x04**

Bit	Attribute	Default	Description
31			Reserved.
30	RW	0	Interrupt on Reset Asserted and waiting for SW to proceed enable. When set, the IRQ is set when the sequencer is waiting for the software to proceed in an assertion sequence.
29	RW	0	Interrupt on Reset Deasserted and waiting for SW to proceed enable. When set, the IRQ is set when the sequencer is waiting for the software to proceed in a deassertion sequence.
28:26			Reserved.
25:16	RW	0	Interrupt on Reset deassertion input qualification signal <code>reset_dsrt_qual_[9:0]</code> status— When set, the IRQ is set when the <code>reset_dsrt_qual[9:0]</code> status bit (per bit enable) is set.
15:12			Reserved.
11	RW	0	Interrupt on <code>reset_in9</code> Enable—When set, the IRQ is set when the <code>reset_in9</code> trigger status bit is set.
10	RW	0	Interrupt on <code>reset_in8</code> Enable—When set, the IRQ is set when the <code>reset_in8</code> trigger status bit is set.
9	RW	0	Interrupt on <code>reset_in7</code> Enable—When set, the IRQ is set when the <code>reset_in7</code> trigger status bit is set.
8	RW	0	Interrupt on <code>reset_in6</code> Enable—When set, the IRQ is set when the <code>reset_in6</code> trigger status bit is set.
7	RW	0	Interrupt on <code>reset_in5</code> Enable—When set, the IRQ is set when the <code>reset_in5</code> trigger status bit is set.
6	RW	0	Interrupt on <code>reset_in4</code> Enable—When set, the IRQ is set when the <code>reset_in4</code> trigger status bit is set.
5	RW	0	Interrupt on <code>reset_in3</code> Enable—When set, the IRQ is set when the <code>reset_in3</code> trigger status bit is set.
4	RW	0	Interrupt on <code>reset_in2</code> Enable—When set, the IRQ is set when the <code>reset_in2</code> trigger status bit is set.
3	RW	0	Interrupt on <code>reset_in1</code> Enable—When set, the IRQ is set when the <code>reset_in1</code> trigger status bit is set.
2	RW	0	Interrupt on <code>reset_in0</code> Enable—When set, the IRQ is set when the <code>reset_in0</code> trigger status bit is set.
1	RW	0	Interrupt on Software triggered reset Enable—When set, the IRQ is set when the software triggered reset status bit is set.
0	RW	0	Interrupt on Power-On-Reset Enable—When set, the IRQ is set when the power-on-reset status bit is set.

#### Related Links

[Reset Sequencer CSR Registers on page 707](#)

### 11.5.4.3.3 Reset Sequencer Control Register

The `Control` register allows you to control the Reset Sequencer.

**Table 164. Values for the Control Register at Offset 0x08**

Bit	Attribute	Default	Description
31:3			Reserved.
2	RW	0	Enable SW sequenced reset assert—Enable a software sequenced reset assert sequence. Timer delays and input qualification are ignored, and only the software can sequence the assert.
1	RW	0	Enable SW sequenced reset deassert—Enable a software sequenced reset deassert sequence. Timer delays and input qualification are ignored, and only the software can sequence the deassert.
0	WO	0	Initiate Reset Sequence—To trigger the hardware sequenced warm reset, the Reset Sequencer writes this bit to 1 a single time. The Reset Sequencer verifies that <code>Reset Active</code> is 0 before setting this bit, and always reads the value 0. To monitor this sequence, verify that <code>Reset Active</code> is asserted, and then subsequently deasserted.

#### Related Links

[Reset Sequencer CSR Registers](#) on page 707

### 11.5.4.3.4 Reset Sequencer Software Sequenced Reset Assert Control Register

You can program the `Software Sequenced Reset Assert` control register to control the reset assertion sequence.

When the corresponding enable bit is set, the sequencer stops when the desired reset asserts, and then sets the `Reset Asserted` and waiting for SW to proceed bit. The Reset Sequencer proceeds only after the `Reset Asserted` and waiting for SW to proceed bit is cleared.

**Table 165. Values for the Reset Sequencer Software Sequenced Reset Assert Control Register at Offset 0x0C**

Bit	Attribute	Default	Description
31:10			Reserved.
9:0	RW	0x3FF	Per-reset SW sequenced reset assert enable—This is a per-bit enable for SW sequenced reset assert. If the register's <code>bitN</code> is set, the sequencer sets the <code>bit30</code> of the status register when a <code>resetN</code> is asserted. It then waits for the <code>bit30</code> of the status register to clear before proceeding with the sequence. By default, all bits are enabled (fully SW sequenced).

#### Related Links

[Reset Sequencer CSR Registers](#) on page 707

### 11.5.4.3.5 Reset Sequencer Software Sequenced Reset Deassert Control Register

You can program the `Software Sequenced Reset Deassert` register to control the reset deassertion sequence.



When the corresponding enable bit is set, the sequencer stops when the desired reset asserts, and then sets the Reset Deasserted and waiting for SW to proceed bit. The Reset Sequencer proceeds only after the Reset Deasserted and waiting for SW to proceed bit is cleared.

**Table 166. Values for the Reset Sequencer Software Sequenced Reset Deassert Control Register at Offset 0x10**

Bit	Attribute	Default	Description
31:10	Reserved.		
9:0	RW	0x3FF	Per-reset SW sequenced reset deassert enable—This is a per-bit enable for SW-sequenced reset deassert. If bitN of this register is set, the sequencer sets bit29 of the Status Register when a resetN is asserted. It then waits for the bit29 of the status register to clear before proceeding with the sequence. By default, all bits are enabled (fully SW sequenced).

**Related Links**

[Reset Sequencer CSR Registers on page 707](#)

**11.5.4.3.6 Reset Sequencer Software Direct Controlled Resets**

You can write a bit to 1 to assert the reset\_outN signal, and to 0 to deassert the reset\_outN signal.

**Table 167. Values for the Software Direct Controlled Resets at Offset 0x14**

Bit	Attribute	Default	Description
31:26	Reserved.		
25:16	WO	0	Reset Overwrite Trigger Enable—This is a per-bit control trigger bit for the overwrite value to take effect.
15:10	Reserved.		
9:0	WO	0	reset_outN Reset Overwrite Value—This is a per-bit control of the reset_out bit. The Reset Sequencer can use this to forcefully drive the reset to a specific value. A value of 1 sets the reset_out. A value of 0 clears the reset_out. A write to this register only takes effect if the corresponding trigger bit in this register is set.

**Related Links**

[Reset Sequencer CSR Registers on page 707](#)

**11.5.4.3.7 Reset Sequencer Software Reset Masking**

Masking off (writing 1) to a reset\_outN "Reset Mask Enable" signal prevents the corresponding reset from being asserted. Writing a bit to 0 to a reset mask enable signal allows assertion of reset\_outN.



**Table 168. Values for the Reset Sequencer Software Reset Masking at Offset 0x18**

Bit	Attribute	Default	Description
31:10			Reserved.
9:0	RW	0	reset_outN "Reset Mask Enable"—This is a per-bit control to mask off the reset_outN bit. Software Reset Masking prevents the reset bit from being asserted during a reset assertion sequence. If reset_out is already asserted, it does not deassert the reset.

**Related Links**

[Reset Sequencer CSR Registers on page 707](#)

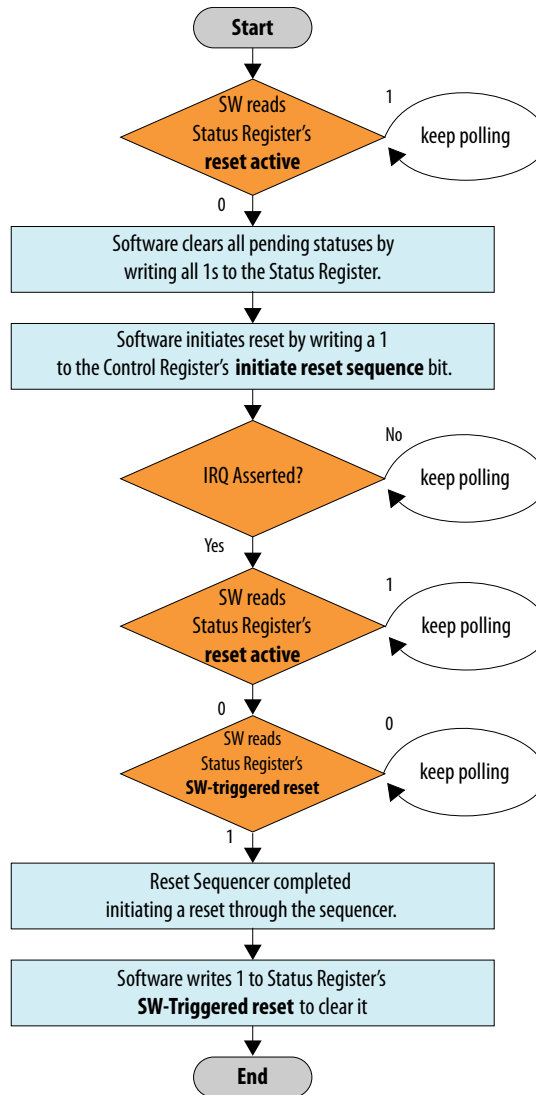




## 11.5.4.4 Reset Sequencer Software Flows

### 11.5.4.4.1 Reset Sequencer (Software-Triggered) Flow

Figure 239. Reset Sequencer (Software-Triggered) Flow Diagram



#### Related Links

- [Reset Sequencer Status Register](#) on page 707
- [Reset Sequencer Control Register](#) on page 710

#### 11.5.4.4.2 Reset Assert Flow

The following flow sequence occurs for a Reset Assert Flow:

- A reset is triggered either by the software, or when input resets to the Reset Sequencer are asserted.
- The IRQ is asserted, if the IRQ is enabled.
- Software reads the Status register to determine which reset was triggered.

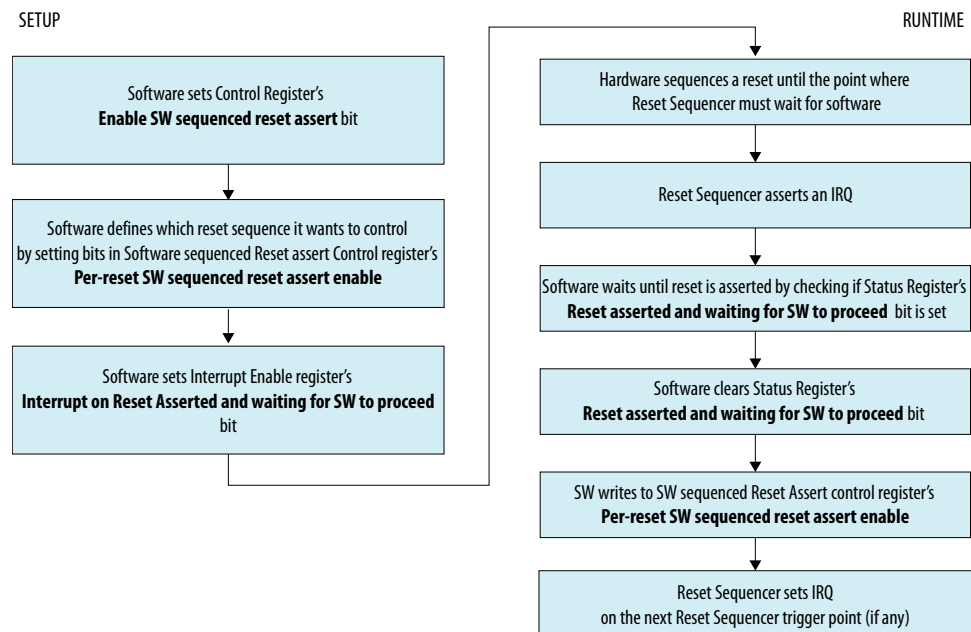
#### 11.5.4.4.3 Reset Deassert Flow

The following flow sequence occurs for a Reset Deassert Flow:

- When a reset source is deasserted, or when the reset assert sequence has completed without pending resets asserted, the deassertion flow is initiated.
- The IRQ is asserted, if the IRQ is enabled.
- Software reads the Status Register to determine which reset was triggered.

#### 11.5.4.4.4 Reset Assert (Software Sequenced) Flow

Figure 240. Reset Assert (Software Sequenced) Flow



#### Related Links

- [Reset Sequencer Control Register](#) on page 710
- [Reset Sequencer Software Sequenced Reset Assert Control Register](#) on page 710
- [Reset Sequencer Interrupt Enable Register](#) on page 709
- [Reset Sequencer Status Register](#) on page 707



#### 11.5.4.4.5 Reset Deassert (Software Sequenced) Flow

The sequence and flow is similar to the Reset Assert (SW Sequenced) flow, though, this flow uses the `reset_deassert` registers/bits instead of the `reset_assert` registers/bits.

##### Related Links

[Reset Assert \(Software Sequenced\) Flow](#) on page 714

## 11.6 Conduits

You can use the conduit interface type for interfaces that do not fit any of the other interface types, and to group any arbitrary collection of signals. Like other interface types, you can export or connect conduit interfaces.

The PCI Express-to-Ethernet example in *Creating a System with Platform Designer* is an example of using a conduit interface for export. You can declare an associated clock interface for conduit interfaces in the same way as memory-mapped interfaces with the `associatedClock`.

To connect two conduit interfaces inside Platform Designer, the following conditions must be met:

- The interfaces must match exactly with the same signal roles and widths.
- The interfaces must be the opposite directions.
- Clocked conduit connections must have matching `associatedClocks` on each of their endpoint interfaces.

*Note:* To connect a conduit output to more than one input conduit interface, you can create a custom component. The custom component could have one input that connects to two outputs, and you can use this component between other conduits that you want to connect. For information about the Avalon Conduit interface, refer to the *Avalon Interface Specifications*

##### Related Links

- [Creating a System with Platform Designer](#) on page 327
- [Avalon Interface Specifications](#)

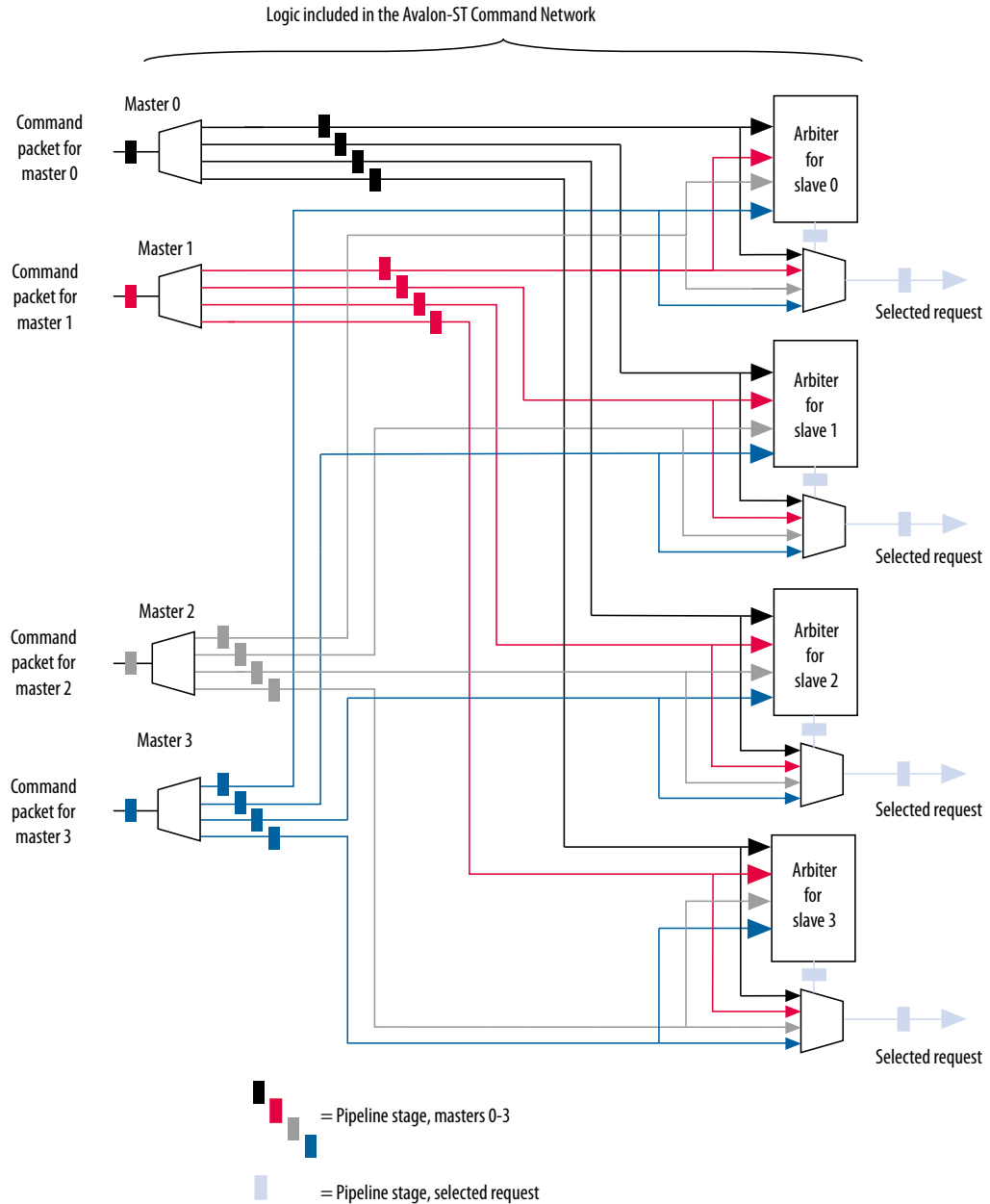
## 11.7 Interconnect Pipelining

Platform Designer can automatically insert Avalon-ST pipeline stages when you generate your design. To do so, set the **Limit interconnect pipeline stages to** parameter to a value greater than 0 in the **Project Settings** tab. The pipeline stages increase the  $f_{MAX}$  of your design by reducing the combinational logic depth. The cost is additional latency and logic.

The insertion of pipeline stages depends upon the existence of certain interconnect components. For example, in a single-slave system, no multiplexer exists; therefore, multiplexer pipelining does not occur. In an extreme case, of a single-master to single-slave system, no pipelining occurs, regardless of the value of the **Limit interconnect pipeline stages to** option.

**Figure 241. Pipeline Placement in Arbitration Logic**

The example below shows the possible placement of up to four potential pipeline stages, which could be, before the input to the demultiplexer, at the output of the multiplexer, between the arbiter and the multiplexer, and at the outputs of the demultiplexer.



**Related Links**

- [Explore and Manage Platform Designer Interconnect](#) on page 391
- [Inserting Pipeline Stages to Increase System Frequency](#) on page 753



## 11.7.1 Manually Controlling Pipelining in the Platform Designer Interconnect

The **Memory-Mapped Interconnect** tab allows you to manipulate pipeline connections in the Platform Designer interconnect. Access the **Memory-Mapped Interconnect** tab by clicking **System > Show System With Platform Designer Interconnect**

*Note:*

To increase interconnect frequency, you should first try increasing the value of the **Limit interconnect pipeline stages to** option on the **Interconnect Requirements** tab. You should only consider manually pipelining the interconnect if changes to this option do not improve frequency, and you have tried all other options to achieve timing closure, including the use of a bridge. Manually pipelining the interconnect should only be applied to complete systems.

1. In the **Interconnect Requirements** tab, first try increasing the value of the **Limit interconnect pipeline stages to** option until it no longer gives significant improvements in frequency, or until it causes unacceptable effects on other parts of the system.
2. In the Intel Quartus Prime software, compile your design and run timing analysis.
3. Using the timing report, identify the critical path through the interconnect and determine the approximate mid-point. The following is an example of a timing report:

```
2.800 0.000 cpu_instruction_master|out_shifter[63]|q
3.004 0.204 mm_domain_0|addr_router_001|Equal5~0|datac
3.246 0.242 mm_domain_0|addr_router_001|Equal5~0|combout
3.346 0.100 mm_domain_0|addr_router_001|Equal5~1|dataa
3.685 0.339 mm_domain_0|addr_router_001|Equal5~1|combout
4.153 0.468 mm_domain_0|addr_router_001|src_channel[5]~0|datad
4.373 0.220 mm_domain_0|addr_router_001|src_channel[5]~0|combout
```

4. In Platform Designer, click **System > Show System With Platform Designer Interconnect**.
5. In the **Memory-Mapped Interconnect** tab, select the interconnect module that contains the critical path. You can determine the name of the module from the hierarchical node names in the timing report.
6. Click **Show Pipelinable Locations**. Platform Designer display all possible pipeline locations in the interconnect. Right-click the possible pipeline location to insert or remove a pipeline stage.
7. Locate the possible pipeline location that is closest to the mid-point of the critical path. The names of the blocks in the memory-mapped interconnect tab correspond to the module instance names in the timing report.
8. Right-click the location where you want to insert a pipeline, and then click **Insert Pipeline**.
9. Regenerate the Platform Designer system, recompile the design, and then rerun timing analysis. If necessary, repeat the manual pipelining process again until timing requirements are met.

Manual pipelining has the following limitations:

- If you make changes to your original system's connectivity after manually pipelining an interconnect, your inserted pipelines may become invalid. Platform Designer displays warning messages when you generate your system if invalid pipeline stages are detected. You can remove invalid pipeline stages with the **Remove Stale Pipelines** option in the **Memory-Mapped Interconnect** tab. Intel recommends that you do not make changes to the system's connectivity after manual pipeline insertion.
- Review manually-inserted pipelines when upgrading to newer versions of Platform Designer. Manually-inserted pipelines in one version of Platform Designer may not be valid in a future version.

**Related Links**

[Specify Platform Designer Interconnect Requirements](#) on page 371

## 11.8 Error Correction Coding (ECC) in Platform Designer Interconnect

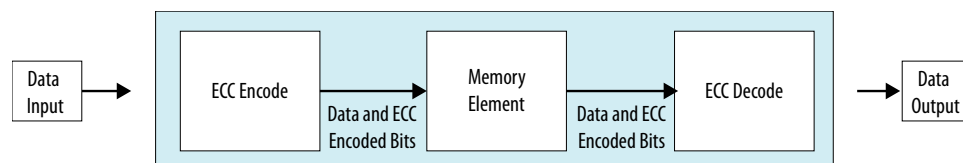
Error Correction Coding (ECC) allows the Platform Designer interconnect to detect and correct errors in order to improve data integrity in memory blocks.

As transistors become smaller, computer hardware is more susceptible to data corruption. Data corruption causes Single Event Upsets (SEUs) and increases the probability of Failures in Time (FIT) rates in computer systems. SEU events without error notification can cause the system to be stuck in an unknown response state, and increase the probability of FIT rates.

ECC encodes the data bus with a Hamming code before it writes it to the memory device, and then decodes and performs error checking on the data on output.

*Note:* Platform Designer sends uncorrectable errors in memory elements as a DECERR on the response bus. This feature is currently only supported for `rdata_FIFO` instances when back pressure occurs on the `wait_request` signal.

**Figure 242. High-Level Implementation of RDATA FIFO with ECC Enabled**



**Related Links**

[Read and Write Responses](#) on page 681

## 11.9 AMBA 3 AXI Protocol Specification Support (version 1.0)

Platform Designer allows memory-mapped connections between AMBA 3 AXI components, AMBA 3 AXI and AMBA 4 AXI components, and AMBA 3 AXI and Avalon interfaces with unique or exceptional support. Refer to the *AMBA 3 Protocol Specifications* on the ARM website for more information.



### Related Links

- [AMBA 3 Protocol Specifications](#)
- [Slave Network Interfaces](#) on page 669

## 11.9.1 Channels

Platform Designer has the following support and restrictions for AMBA 3 AXI channels.

### 11.9.1.1 Read and Write Address Channels

Most signals are allowed. However, the following limitations are present in Platform Designer 14.0:

- Supports 64-bit addressing.
- ID width limited to 18-bits.
- HPS-FPGA master interface has a 12-bit ID.

### 11.9.1.2 Write Data, Write Response, and Read Data Channels

Most signals are allowed. However, the following limitations are present in Platform Designer 14.0:

- Data widths limited to a maximum of 1024-bits
- Limited to a fixed byte width of 8-bits

### 11.9.1.3 Low Power Channel

Low power extensions are not supported in Platform Designer, version 14.0.

## 11.9.2 Cache Support

AWCACHE and ARCACHE are passed to an AXI slave unmodified.

### 11.9.2.1 Bufferable

Platform Designer interconnect treats AXI transactions as non-bufferable. All responses must come from the terminal slave.

When connecting to Avalon-MM slaves, since they do not have write responses, the following exceptions apply:

- For Avalon-MM slaves, the write response are generated by the slave agent once the write transaction is accepted by the slave. The following limitation exists for an Avalon bridge:
- For an Avalon bridge, the response is generated before the write reaches the endpoint; users must be aware of this limitation and avoid multiple paths past the bridge to any endpoint slave, or only perform bufferable transactions to an Avalon bridge.

### 11.9.2.2 Cacheable (Modifiable)

Platform Designer interconnect acknowledges the cacheable (modifiable) attribute of AXI transactions.



It does not change the address, burst length, or burst size of non-modifiable transactions, with the following exceptions:

- Platform Designer considers a wide transaction to a narrow slave as modifiable because the size requires reduction.
- Platform Designer may consider AXI read and write transactions as modifiable when the destination is an Avalon slave. The AXI transaction may be split into multiple Avalon transactions if the slave is unable to accept the transaction. This may occur because of burst lengths, narrow sizes, or burst types.

Platform Designer ignores all other bits, for example, read allocate or write allocate because the interconnect does not perform caching. By default, Platform Designer considers Avalon master transactions as non-bufferable and non-cacheable, with the allocate bits tied low.

### 11.9.3 Security Support

TrustZone refers to the security extension of the ARM architecture, which includes the concept of "secure" and "non-secure" transactions, and a protocol for processing between the designations.

The interconnect passes the `AWPROT` and `ARPROT` signals to the endpoint slave without modification. It does not use or modify the `PROT` bits.

Refer to *Manage System Security in Creating a System with Platform Designer* for more information about secure systems and the TrustZone feature.

#### Related Links

[Manage Platform Designer System Security](#) on page 373

### 11.9.4 Atomic Accesses

Exclusive accesses are supported for AXI slaves by passing the lock, transaction ID, and response signals from master to slave, with the limitation that slaves that do not reorder responses. Avalon slaves do not support exclusive accesses, and always return `OKAY` as a response. Locked accesses are also not supported.

### 11.9.5 Response Signaling

Full response signaling is supported. Avalon slaves always return `OKAY` as a response.

### 11.9.6 Ordering Model

Platform Designer interconnect provides responses in the same order as the commands are issued.

To prevent reordering, for slaves that accept reordering depths greater than 0, Platform Designer does not transfer the transaction ID from the master, but provides a constant transaction ID of 0. For slaves that do not reorder, Platform Designer allows the transaction ID to be transferred to the slave. To avoid cyclic dependencies, Platform Designer supports a single outstanding slave scheme for both reads and writes. Changing the targeted slave before all responses have returned stalls the master, regardless of transaction ID.





### 11.9.6.1 AXI and Avalon Ordering

There is a potential read-after-write risk when Avalon masters transact to AXI slaves.

According to the *AMBA Protocol Specifications*, there is no ordering requirement between reads and writes. However, Avalon has an implicit ordering model that requires transactions from a master to the same slave to be in order.

In response to this potential risk, Avalon interfaces provide a compile-time option to enforce strict order. When turned on, the Avalon interface waits for outstanding write responses before issuing reads.

### 11.9.7 Data Buses

Narrow bus transfers are supported. AXI write strobes can have any pattern that is compatible with the address and size information. Intel recommends that transactions to Avalon slaves follow Avalon `byteenable` limitations for maximum compatibility.

*Note:* Byte 0 is always bits [7:0] in the interconnect, following AXI's and Avalon's byte (address) invariance scheme.

### 11.9.8 Unaligned Address Commands

Unaligned address commands are commands with addresses that do not conform to the data width of a slave. Since Avalon-MM slaves accept only aligned addresses, Platform Designer modifies unaligned commands from AXI masters to the correct data width. Platform Designer must preserve commands issued by AXI masters when passing the commands to AXI slaves.

*Note:* Unaligned transfers are aligned if downsizing occurs. For example, when downsizing to a bus width narrower than that required by the transaction size, `AWSIZE` or `ARSIZE`, the transaction must be modified.

### 11.9.9 Avalon and AXI Transaction Support

Platform Designer 14.0 supports transactions between Avalon and interfaces, with some limitations.

#### 11.9.9.1 Transaction Cannot Cross 4KB Boundaries

When an Avalon master issues a transaction to an AXI slave, the transaction cannot cross 4KB boundaries. Non-bursting Avalon masters already follow this boundary restriction.

#### 11.9.9.2 Handling Read Side Effects

Read side effects can occur when more bytes than necessary are read from the slave, and the unwanted data that are read are later inaccessible on subsequent reads. For write commands, the correct `byteenable` paths are asserted based on the size of the transactions. For read commands, narrow-sized bursts are broken up into multiple non-bursting commands, and each command with the correct `byteenable` paths asserted.

Platform Designer always assumes that the byteenable is asserted based on the size of the command, not the address of the command. The following scenarios are examples:

- For a 32-bit AXI master that issues a read command with an unaligned address starting at address `0x01`, and a burstcount of 2 to a 32-bit Avalon slave, the starting address is: `0x00`.
- For a 32-bit AXI master that issues a read command with an unaligned address starting at address `0x01`, with 4-bytes to an 8-bit AXI slave, the starting address is: `0x00`.

## 11.10 AMBA 3 APB Protocol Specification Support (version 1.0)

APB (Advanced Peripheral Bus) interface is optimized for minimal power consumption and reduced interface complexity. You can use APB to interface to peripherals which are low-bandwidth and do not require the high performance of a pipelined bus interface. Signal transitions are sampled at the rising edge of the clock to enable the integration of APB peripherals easily into any design flow.

Platform Designer allows connections between APB components, and AMBA 3 AXI, AMBA 4 AXI, and Avalon memory-mapped interfaces. The following sections describe unique or exceptional APB support in the Platform Designer software.

### Related Links

[Arm AMBA Protocol Specifications](#)

### 11.10.1 Bridges

With APB, you cannot use bridge components that use multiple `PSELx` in Platform Designer. As a workaround, you can group `PSELx`, and then send the packet to the slave directly.

Intel recommends as an alternative that you instantiate the APB bridge and all the APB slaves in Platform Designer. You should then connect the slave side of the bridge to any high speed interface and connect the master side of the bridge to the APB slaves. Platform Designer creates the interconnect on either side of the APB bridge and creates only one `PSEL` signal.

Alternatively, you can connect a bridge to the APB bus outside of Platform Designer. Use an Avalon/AXI bridge to export the Avalon/AXI master to the top-level, and then connect this Avalon/AXI interface to the slave side of the APB bridge. Alternatively, instantiate the APB bridge in Platform Designer and export APB master to the top-level, and from there connect to APB bus outside of Platform Designer.

### 11.10.2 Burst Adaptation

APB is a non-bursting interface. Therefore, for any AXI or Avalon master with bursting support, a burst adapter is inserted before the slave interface and the burst transaction is translated into a series of non-bursting transactions before reaching the APB slave.



### 11.10.3 Width Adaptation

Platform Designer allows different data width connections with APB. When connecting a wider master to a narrower APB slave, the width adapter converts the wider transactions to a narrower transaction to fit the APB slave data width. APB does not support Write Strobe. Therefore, when you connect a narrower transaction to a wider APB slave, the slave cannot determine which byte lane to write. In this case, the slave data may be overwritten or corrupted.

### 11.10.4 Error Response

Error responses are returned to the master. Platform Designer performs error mapping if the master is an AMBA 3 AXI or AMBA 4 AXI master, for example, `RRESP/BRESP=SLVERR`. For the case when the slave does not use `SLVERR` signal, an `OKAY` response is sent back to master by default.

## 11.11 AMBA 4 AXI Memory-Mapped Interface Support (version 2.0)

Platform Designer allows memory-mapped connections between AMBA 4 AXI components, AMBA 4 AXI and AMBA 3 AXI components, and AMBA 4 AXI and Avalon interfaces with unique or exceptional support.

### 11.11.1 Burst Support

Platform Designer supports `INCR` bursts up to 256 beats. Platform Designer converts long bursts to multiple bursts in a packet with each burst having a length less than or equal to `MAX_BURST` when going to AMBA 3 AXI or Avalon slaves.

For narrow-sized transfers, bursts with Avalon slaves as destinations are shortened to multiple non-bursting transactions in order to transmit the correct address to the slaves, since Avalon slaves always perform full-sized `datawidth` transactions.

Bursts with AMBA 3 AXI slaves as destinations are shortened to multiple bursts, with each burst length less than or equal to 16. Bursts with AMBA 4 AXI slaves as destinations are not shortened.

### 11.11.2 QoS

Platform Designer routes 4-bit QoS signals (Quality of Service Signaling) on the read and write address channels directly from the master to the slave.

Transactions from AMBA 3 AXI and Avalon masters have a default value of `4'b0000`, which indicates that the transactions are not part of the QoS flow. QoS values are not used for slaves that do not support QoS.

For Platform Designer 14.0, there are no programmable QoS registers or compile-time QoS options for a master that overrides its real or default value.

### 11.11.3 Regions

For Platform Designer 14.0, there is no support for the optional regions feature. AMBA 4 AXI slaves with `AXREGION` signals are allowed. `AXREGION` signals are driven with the default value of `0x0`, and are limited to one entry in a master's address map.



### 11.11.4 Write Response Dependency

Write response dependency as specified in the *Arm AMBA Protocol Specifications* for AMBA 4 AXI is not supported.

#### Related Links

[Arm AMBA Protocol Specifications](#)

### 11.11.5 AWCACHE and ARCACHE

For AMBA 4 AXI, Platform Designer meets the requirement for modifiable and non-modifiable transactions. The modifiable bit refers to `ARCACHE[1]` and `AWCACHE[1]`.

### 11.11.6 Width Adaptation and Data Packing in Platform Designer

Data packing applies only to systems where the data width of masters is less than the data width of slaves.

The following rules apply:

- Data packing is supported when masters and slaves are Avalon-MM.
- Data packing is not supported when any master or slave is an AMBA 3 AXI, AMBA 4 AXI, or APB component.

For example, for a read/write command with a 32-bit master connected to a 64-bit slave, and a transaction of 2 burstcounts, Platform Designer sends 2 separate read/write commands to access the 64-bit data width of the slave. Data packing is only supported if the system does not contain AMBA 3 AXI, AMBA 4 AXI, or APB masters or slaves.

### 11.11.7 Ordering Model

Out of order support is not implemented in Platform Designer, version 14.0. Platform Designer processes AXI slaves as device non-bufferable memory types.

The following describes the required behavior for the device non-bufferable memory type:

- Write response must be obtained from the final destination.
- Read data must be obtained from the final destination.
- Transaction characteristics must not be modified.
- Reads must not be pre-fetched. Writes must not be merged.
- Non-modifiable read and write transactions.

(`AWCACHE[1] = 0` or `ARCACHE[1] = 0`) from the same ID to the same slave must remain ordered. The interconnect always provides responses in the same order as the commands issued. Slaves that support reordering provide a constant transaction ID to prevent reordering. AXI slaves that do not reorder are provided with transaction IDs, which allows exclusive accesses to be used for such slaves.



### 11.11.8 Read and Write Allocate

Read and write allocate does not apply to Platform Designer interconnect, which does not have caching features, and always receives responses from an endpoint.

### 11.11.9 Locked Transactions

Locked transactions are not supported for Platform Designer, version 14.0.

### 11.11.10 Memory Types

For AMBA 4 AXI, Platform Designer processes transactions as though the endpoint is a device memory type. For device memory types, using non-bufferable transactions to force previous bufferable transactions to finish is irrelevant, because Platform Designer interconnect always identifies transactions as being non-bufferable.

### 11.11.11 Mismatched Attributes

There are rules for how multiple masters issue cache values to a shared memory region. The interconnect meets requirements if signals are not modified.

### 11.11.12 Signals

Platform Designer supports up to 64-bits for the BUSER, WUSER and RUSER sideband signals. AMBA 4 AXI allows some signals to be omitted from interfaces by aligning them with the default values as defined in the *AMBA Protocol Specifications* on the ARM website.

#### Related Links

[Arm AMBA Protocol Specifications](#)

## 11.12 AMBA 4 AXI Streaming Interface Support (version 1.0)

### 11.12.1 Connection Points

Platform Designer allows you to connect an AMBA 4 AXI-Stream interface to another AMBA 4 AXI-Stream interface.

The connection is point-to-point without adaptation and must be between an `axi4stream_master` and `axi4stream_slave`. Connected interfaces must have the same port roles and widths.

Non matching master to slave connections, and multiple masters to multiple slaves connections are not supported.

#### 11.12.1.1 AMBA 4 AXI Streaming Connection Point Parameters

**Table 169. AMBA 4 AXI Streaming Connection Point Parameters**

Name	Type	Description
associatedClock	string	Name of associated clock interface.
associatedReset	string	Name of associated reset interface

### 11.12.1.2 AMBA 4 AXI Streaming Connection Point Signals

**Table 170. AMBA 4 AXI-Stream Connection Point Signals**

Port Role	Width	Master Direction	Slave Direction	Required
tvalid	1	Output	Input	Yes
tready	1	Input	Output	No
tdata <sup>(9)</sup>	8:4096	Output	Input	No
tstrb	1:512	Output	Input	No
tkeep	1:512	Output	Input	No
tid <sup>(10)</sup>	1:8	Output	Input	No
tdest <sup>(11)</sup>	1:4	Output	Input	No
tuser <sup>(12)</sup>	1:4096	Output	Input	No
tlast	1	Output	Input	No

### 11.12.2 Adaptation

AMBA 4 AXI-Stream adaptation support is not available. AMBA 4 AXI-Stream master and slave interface signals and widths must match.

## 11.13 AMBA 4 AXI-Lite Protocol Specification Support (version 2.0)

AMBA 4 AXI-Lite is a sub-set of AMBA 4 AXI. It is suitable for simpler control register-style interfaces that do not require the full functionality of AMBA 4 AXI.

Platform Designer 14.0 supports the following AMBA 4 AXI-Lite features:

- Transactions with a burst length of 1.
- Data accesses use the full width of a data bus (32-bit or 64-bit) for data accesses, and no narrow-size transactions.
- Non-modifiable and non-bufferable accesses.
- No exclusive accesses.

<sup>(9)</sup> integer in mutiple of bytes

<sup>(10)</sup> maximum 8-bits

<sup>(11)</sup> maximum 4-bits

<sup>(12)</sup> number of bits in multiple of the number of bytes of tdata



### 11.13.1 AMBA 4 AXI-Lite Signals

Platform Designer supports all AMBA 4 AXI-Lite interface signals. All signals are required.

**Table 171. AMBA 4 AXI-Lite Signals**

Global	Write Address Channel	Write Data Channel	Write Response Channel	Read Address Channel	Read Data Channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
-	AWADDR	WDATA	BRESP	ARADDR	RDATA
-	AWPROT	WSTRB	-	ARPROT	RRESP

### 11.13.2 AMBA 4 AXI-Lite Bus Width

AMBA 4 AXI-Lite masters or slaves must have either 32-bit or 64-bit bus widths. Platform Designer interconnect inserts a width adapter if a master and slave pair have different widths.

### 11.13.3 AMBA 4 AXI-Lite Outstanding Transactions

AXI-Lite supports outstanding transactions. The options to control outstanding transactions is set in the parameter editor for the selected component.

### 11.13.4 AMBA 4 AXI-Lite IDs

AMBA 4 AXI-Lite does not support IDs. Platform Designer performs ID reflection inside the slave agent.

### 11.13.5 Connections Between AMBA 3 AXI, AMBA 4 AXI and AMBA 4 AXI-Lite

#### 11.13.5.1 AMBA 4 AXI-Lite Slave Requirements

For an AMBA 4 AXI-Lite slave side, the master can be any master interface type, such as an Avalon (with bursting), AMBA 3 AXI, or AMBA 4 AXI. Platform Designer allows the following connections and inserts adapters, if needed.

- **Burst adapter**—Avalon and AMBA 3 AXI and AMBA 4 AXI bursting masters require a burst adapter to shorten the burst length to 1 before sending a transaction to an AMBA 4 AXI-Lite slave.
- Platform Designer interconnect uses a width adapter for mismatched data widths.
- Platform Designer interconnect performs ID reflection inside the slave agent.
- An AMBA 4 AXI-Lite slave must have an address width of at least 12-bits.
- AMBA 4 AXI-Lite does not have the `AXSIZE` parameter. Narrow master to a wide AMBA 4 AXI-Lite slave is not supported. For masters that support narrow-sized bursts, for example, AMBA 3 AXI and AMBA 4 AXI, a burst to an AMBA 4 AXI-Lite slave must have a burst size equal to or greater than the slave's burst size.



### 11.13.5.2 AMBA 4 AXI-Lite Data Packing

Platform Designer interconnect does not support AMBA 4 AXI-Lite data packing.

### 11.13.6 AMBA 4 AXI-Lite Response Merging

When Platform Designer interconnect merges SLVERR and DECERR, the error responses are not sticky. The response is based on priority and the master always sees a DECERR. When SLVERR and DECERR are merged, it is based on their priorities, not stickiness. DECERR receives priority in this case, even if SLVERR returns first.

## 11.14 Port Roles (Interface Signal Types)

Each interface defines signal roles and their behavior. Many signal roles are optional, allowing IP component designers the flexibility to select only the signal roles necessary to implement the required functionality.

### 11.14.1 AXI Master Interface Signal Types

**Table 172. AXI Master Interface Signal Types**

Name	Direction	Width
araddr	output	1 - 64
arburst	output	2
arcache	output	4
arid	output	1 - 18
arlen	output	4
arlock	output	2
arprot	output	3
arready	input	1
arsize	output	3
aruser	output	1 - 64
arvalid	output	1
awaddr	output	1 - 64
awburst	output	2
awcache	output	4
awid	output	1 - 18
awlen	output	4
awlock	output	2
awprot	output	3
awready	input	1
awsize	output	3

*continued...*





Name	Direction	Width
awuser	output	1 - 64
awvalid	output	1
bid	input	1 - 18
bready	output	1
bresp	input	2
bvalid	input	1
rdata	input	8, 16, 32, 64, 128, 256, 512, 1024
rid	input	1 - 18
rlast	input	1
rready	output	1
rresp	input	2
rvalid	input	1
wdata	output	8, 16, 32, 64, 128, 256, 512, 1024
wid	output	1 - 18
wlast	output	1
wready	input	1
wstrb	output	1, 2, 4, 8, 16, 32, 64, 128
wvalid	output	1

### 11.14.2 AXI Slave Interface Signal Types

Table 173. AXI Slave Interface Signal Types

Name	Direction	Width
araddr	input	1 - 64
arburst	input	2
arcache	input	4
arid	input	1 - 18
arlen	input	4
arlock	input	2
arprot	input	3
arready	output	1
arsize	input	3
aruser	input	1 - 64
arvalid	input	1
awaddr	input	1 - 64

*continued...*



Name	Direction	Width
awburst	input	2
awcache	input	4
awid	input	1 - 18
awlen	input	4
awlock	input	2
awprot	input	3
awready	output	1
awsize	input	3
awuser	input	1 - 64
awvalid	input	1
bid	output	1 - 18
bready	input	1
bresp	output	2
bvalid	output	1
rdata	output	8, 16, 32, 64, 128, 256, 512, 1024
rid	output	1 - 18
rlast	output	1
rready	input	1
rresp	output	2
rvalid	output	1
wdata	input	8, 16, 32, 64, 128, 256, 512, 1024
wid	input	1 - 18
wlast	input	1
wready	output	1
wstrb	input	1, 2, 4, 8, 16, 32, 64, 128
wvalid	input	1

### 11.14.3 AMBA 4 AXI Master Interface Signal Types

Table 174. AMBA 4 AXI Master Interface Signal Types

Name	Direction	Width
araddr	output	1 - 64
arburst	output	2
arcache	output	4
arid	output	1 - 18
<i>continued...</i>		



Name	Direction	Width
arlen	output	8
arlock	output	1
arprot	output	3
arready	input	1
arregion	output	1 - 4
arsize	output	3
aruser	output	1 - 64
arvalid	output	1
awaddr	output	1 - 64
awburst	output	2
awcache	output	4
awid	output	1 - 18
awlen	output	8
awlock	output	1
awprot	output	3
awqos	output	1 - 4
awready	input	1
awregion	output	1 - 4
awsize	output	3
awuser	output	1 - 64
awvalid	output	1
bid	input	1 - 18
bready	output	1
bresp	input	2
buser	input	1 - 64
bvalid	input	1
rdata	input	8, 16, 32, 64, 128, 256, 512, 1024
rid	input	1 - 18
rlast	input	1
rready	output	1
rresp	input	2
ruser	input	1 - 64
rvalid	input	1
wdata	output	8, 16, 32, 64, 128, 256, 512, 1024
<i>continued...</i>		



Name	Direction	Width
wid	output	1 - 18
wlast	output	1
wready	input	1
wstrb	output	1, 2, 4, 8, 16, 32, 64, 128
wuser	output	1 - 64
wvalid	output	1

### 11.14.4 AMBA 4 AXI Slave Interface Signal Types

Table 175. AMBA 4 AXI Slave Interface Signal Types

Name	Direction	Width
araddr	input	1 - 64
arburst	input	2
arcache	input	4
arid	input	1 - 18
arlen	input	8
arlock	input	1
arprot	input	3
arqos	input	1 - 4
arready	output	1
arregion	input	1 - 4
arsize	input	3
aruser	input	1 - 64
arvalid	input	1
awaddr	input	1 - 64
awburst	input	2
awcache	input	4
awid	input	1 - 18
awlen	input	8
awlock	input	1
awprot	input	3
awqos	input	1 - 4
awready	output	1
awregion	input	1 - 4
awsize	input	3

*continued...*



Name	Direction	Width
awuser	input	1 - 64
awvalid	input	1
bid	output	1 - 18
bready	input	1
bresp	output	2
bvalid	output	1
rdata	output	8, 16, 32, 64, 128, 256, 512, 1024
rid	output	1 - 18
rlast	output	1
rready	input	1
rresp	output	2
ruser	output	1 - 64
rvalid	output	1
wdata	input	8, 16, 32, 64, 128, 256, 512, 1024
wlast	input	1
wready	output	1
wstrb	input	1, 2, 4, 8, 16, 32, 64, 128
wuser	input	1 - 64
wvalid	input	1

### 11.14.5 AMBA 4 AXI-Stream Master and Slave Interface Signal Types

**Table 176. AMBA 4 AXI-Stream Master and Slave Interface Signal Types**

Name	Width	Master Direction	Slave Direction	Required
tvalid	1	Output	Input	Yes
tready	1	Input	Output	No
tdata	8:4096	Output	Input	No
tstrb	1:512	Output	Input	No
tkeep	1:512	Output	Input	No
tid	1:8	Output	Input	No
tdest	1:4	Output	Input	No
tuser	1	Output	Input	No
tlast	1:4096	Output	Input	No



### 11.14.6 APB Interface Signal Types

**Table 177. APB Interface Signal Types**

Name	Width	Direction APB Master	Direction APB Slave	Required
paddr	[1:32]	output	input	yes
pselect	[1:16]	output	input	yes
penable	1	output	input	yes
pwrite	1	output	input	yes
pwrite	[1:32]	output	input	yes
prdata	[1:32]	input	output	yes
pslverr	1	input	output	no
pready	1	input	output	yes
paddr31	1	output	input	no

### 11.14.7 Avalon Memory-Mapped Interface Signal Roles

Signal roles define the signal types that are allowed on Avalon-MM master and slave ports.

This specification does not require all signals to exist in an Avalon-MM interface. There is no one signal that is always required. The minimum requirements for an Avalon-MM interface are `readdata` for a read-only interface, or `writedata` and `write` for a write-only interface.

The following table lists signal roles for the Avalon-MM interface:

**Table 178. Avalon-MM Signal Roles**

Some Avalon-MM signals can be active high or active low. When active low, the signal name ends with `_n`.

Signal Role	Width	Direction	Description
<b>Fundamental Signals</b>			
address	1 - 64	Master <input type="checkbox"/> Slave	Masters: By default, the <code>address</code> signal represents a byte address. The value of the address must be aligned to the data width. To write to specific bytes within a data word, the master must use the <code>byteenable</code> signal. Refer to the <code>addressUnits</code> interface property for word addressing. Slaves: By default, the interconnect translates the byte address into a word address in the slave's address space. Each slave access is for a word of data from the perspective of the slave. For example, <code>address = 0</code> selects the first word of the slave. <code>address = 1</code> selects the second word of the slave. Refer to the <code>addressUnits</code> interface property for byte addressing.
byteenable byteenable_n	2, 4, 8, 16, 32, 64, 128	Master <input type="checkbox"/> Slave	Enables one or more specific byte lanes during transfers on interfaces of width greater than 8 bits. Each bit in <code>byteenable</code> corresponds to a byte in <code>writedata</code> and <code>readdata</code> . The master bit <code>&lt;n&gt;</code> of <code>byteenable</code> indicates whether byte <code>&lt;n&gt;</code> is being written to. During writes, <code>byteenables</code> specify which bytes are being written to.
<i>continued...</i>			



Signal Role	Width	Direction	Description
			<p>Other bytes should be ignored by the slave. During reads, <code>byteenables</code> indicate which bytes the master is reading. Slaves that simply return <code>readdata</code> with no side effects are free to ignore <code>byteenables</code> during reads. If an interface does not have a <code>byteenable</code> signal, the transfer proceeds as if all <code>byteenables</code> are asserted.</p> <p>When more than one bit of the <code>byteenable</code> signal is asserted, all asserted lanes are adjacent. The number of adjacent lines must be a power of 2. The specified bytes must be aligned on an address boundary for the size of the data. For example, the following values are legal for a 32-bit slave:</p> <ul style="list-style-type: none"> <li>• 1111 writes full 32 bits</li> <li>• 0011 writes lower 2 bytes</li> <li>• 1100 writes upper 2 bytes</li> <li>• 0001 writes byte 0 only</li> <li>• 0010 writes byte 1 only</li> <li>• 0100 writes byte 2 only</li> <li>• 1000 writes byte 3 only</li> </ul> <p>To avoid unintended side effects, use the <code>byteenable</code> signal in systems with different word sizes.</p> <p><i>Note:</i> The AXI interface supports unaligned accesses while Avalon-MM does not. Unaligned accesses going from an AXI master to an Avalon-MM slave may result in an illegal transaction. To avoid this issue, only use aligned accesses to Avalon-MM slaves.</p>
debugaccess	1	Master <input type="checkbox"/> Slave	When asserted, allows the Nios II processor to write on-chip memories configured as ROMs.
read read_n	1	Master <input type="checkbox"/> Slave	Asserted to indicate a <code>read</code> transfer. If present, <code>readdata</code> is required.
readdata	8, 16, 32, 64, 128, 256, 512, 1024	Slave <input type="checkbox"/> Master	The <code>readdata</code> driven from the slave to the master in response to a <code>read</code> transfer.
response [1:0]	2	Slave <input type="checkbox"/> Master	<p>The <code>response</code> signal is an optional signal that carries the response status.</p> <p><i>Note:</i> Because the signal is shared, an interface cannot issue or accept a write response and a read response in the same clock cycle.</p> <ul style="list-style-type: none"> <li>• 00: OKAY—Successful response for a transaction.</li> <li>• 01: RESERVED—Encoding is reserved.</li> <li>• 10: SLAVEERROR—Error from an endpoint slave. Indicates an unsuccessful transaction.</li> <li>• 11: DECODEERROR—Indicates attempted access to an undefined location.</li> </ul> <p>For read responses:</p> <ul style="list-style-type: none"> <li>• One response is sent with each <code>readdata</code>. A read burst length of <code>N</code> results in <code>N</code> responses. It is not valid to produce fewer responses, even in the event of an error. It is valid for the response signal value to be different for each <code>readdata</code> in the burst.</li> <li>• The interface must have read control signals. Pipeline support is possible with the <code>readdatavalid</code> signal.</li> <li>• On read errors, the corresponding <code>readdata</code> is "don't care".</li> </ul>

**continued...**



Signal Role	Width	Direction	Description
			<p>For write responses:</p> <ul style="list-style-type: none"> <li>One write response must be sent for each write command. A write burst results in only one response, which must be sent after the final write transfer in the burst is accepted.</li> <li>If <code>writeresponsevalid</code> is present, all write commands must be completed with write responses."</li> </ul>
<code>write</code> <code>write_n</code>	1	Master <input type="checkbox"/> Slave	Asserted to indicate a <code>write</code> transfer. If present, <code>writedata</code> is required.
<code>writedata</code>	8, 16, 32, 64, 128, 256, 512, 1024	Master <input type="checkbox"/> Slave	Data for write transfers. The width must be the same as the width of <code>readdata</code> if both are present.
<b>Wait-State Signals</b>			
<code>lock</code>	1	Master <input type="checkbox"/> Slave	<p><code>lock</code> ensures that once a master wins arbitration, it maintains access to the slave for multiple transactions. It is asserted coincident with the first <code>read</code> or <code>write</code> of a locked sequence of transactions. It is deasserted on the final transaction of a locked sequence of transactions. <code>lock</code> assertion does not guarantee that arbitration is won. After the lock-asserting master has been granted, it retains grant until it is deasserted.</p> <p>A master equipped with <code>lock</code> cannot be a burst master. Arbitration priority values for lock-equipped masters are ignored.</p> <p><code>lock</code> is particularly useful for read-modify-write (RMW) operations. The typical read-modify-write operation includes the following steps:</p> <ol style="list-style-type: none"> <li>Master A asserts <code>lock</code> and reads 32-bit data that has multiple bit fields.</li> <li>Master A deasserts <code>lock</code>, changes one bit field, and writes the 32-bit data back.</li> </ol> <p><code>lock</code> prevents master B from performing a write between Master A's read and write.</p>
<code>waitrequest</code> <code>waitrequest_n</code>	1	Slave <input type="checkbox"/> Master	<p>Asserted by the slave when it is unable to respond to a <code>read</code> or <code>write</code> request. Forces the master to wait until the interconnect is ready to proceed with the transfer. At the start of all transfers, a master initiates the transfer and waits until <code>waitrequest</code> is deasserted. A master must make no assumption about the assertion state of <code>waitrequest</code> when the master is idle: <code>waitrequest</code> may be high or low, depending on system properties.</p> <p>When <code>waitrequest</code> is asserted, master control signals to the slave must remain constant with the exception of <code>beginbursttransfer</code>. For a timing diagram illustrating the <code>beginbursttransfer</code> signal, refer to the figure in <i>Read Bursts</i>.</p> <p>An Avalon-MM slave may assert <code>waitrequest</code> during idle cycles. An Avalon-MM master may initiate a transaction when <code>waitrequest</code> is asserted and wait for that signal to be deasserted. To avoid system lockup, a slave device should assert <code>waitrequest</code> when in reset.</p>
<b>Pipeline Signals</b>			
<code>readdatavalid</code> <code>readdatavalid_n</code>	1	Slave <input type="checkbox"/> Master	Used for variable-latency, pipelined <code>read</code> transfers. When asserted, indicates that the <code>readdata</code> signal contains valid data. For a read burst with <code>burstcount</code> value $\langle n \rangle$ , the <code>readdatavalid</code> signal must be asserted $\langle n \rangle$ times, once for each <code>readdata</code> item. There must be at least one cycle of
<i>continued...</i>			





Signal Role	Width	Direction	Description
			<p>latency between acceptance of the <code>read</code> and assertion of <code>readdatavalid</code>. For a timing diagram illustrating the <code>readdatavalid</code> signal, refer to <i>Pipelined Read Transfer with Variable Latency</i>.</p> <p>A slave may assert <code>readdatavalid</code> to transfer data to the master independently of whether or not the slave is stalling a new command with <code>waitrequest</code>.</p> <p>Required if the master supports pipelined reads. Bursting masters with read functionality must include the <code>readdatavalid</code> signal.</p>
<code>writeresponsevalid</code>			<p>An optional signal. If present, the interface issues write responses for write commands.</p> <p>When asserted, the value on the response signal is a valid write response.</p> <p><code>Writeresponsevalid</code> is only asserted one clock cycle or more after the write command is accepted. There is at least a one clock cycle latency from command acceptance to assertion of <code>writeresponsevalid</code>.</p>
<b>Burst Signals</b>			
<code>burstcount</code>	1 - 11	Master <input type="checkbox"/> Slave	<p>Used by bursting masters to indicate the number of transfers in each burst. The value of the maximum <code>burstcount</code> parameter must be a power of 2. A <code>burstcount</code> interface of width <code>&lt;n&gt;</code> can encode a max burst of size <math>2^{(&lt;n&gt;-1)}</math>. For example, a 4-bit <code>burstcount</code> signal can support a maximum burst count of 8. The minimum <code>burstcount</code> is 1. The <code>constantBurstBehavior</code> property controls the timing of the <code>burstcount</code> signal. Bursting masters with read functionality must include the <code>readdatavalid</code> signal.</p> <p>For bursting masters and slaves using byte addresses, the following restriction applies to the width of the address:</p> <pre style="background-color: #f0f0f0; padding: 5px;"> &lt;address_w&gt; &gt;= &lt;burstcount_w&gt; + log<sub>2</sub>(&lt;symbols_per_word_of_interface&gt;) </pre> <p>For bursting masters and slaves using word addresses, the <math>\log_2</math> term above is omitted.</p>
<code>beginbursttransfer</code>	1	Interconnect <input type="checkbox"/> Slave	<p>Asserted for the first cycle of a burst to indicate when a burst transfer is starting. This signal is deasserted after one cycle regardless of the value of <code>waitrequest</code>. For a timing diagram illustrating <code>beginbursttransfer</code>, refer to the figure in <i>Read Bursts</i>.</p> <p><code>beginbursttransfer</code> is optional. A slave can always internally calculate the start of the next write burst transaction by counting data transfers.</p> <p><b>Warning:</b> <i>do not</i> use this signal. This signal exists to support legacy memory controllers.</p>

### 11.14.8 Avalon Streaming Interface Signal Roles

Each signal in an Avalon-ST source or sink interface corresponds to one Avalon-ST signal role. An Avalon-ST interface may contain only one instance of each signal role. All Avalon-ST signal roles apply to both sources and sinks and have the same meaning for both.

**Table 179. Avalon-ST Interface Signals**

In the following table, all signal roles are active high.

Signal Role	Width	Direction	Description
<b>Fundamental Signals</b>			
channel	1 – 128	Source □ Sink	The channel number for data being transferred on the current cycle. If an interface supports the channel signal, it must also define the maxChannel parameter.
data	1 – 4,096	Source □ Sink	The data signal from the source to the sink, typically carries the bulk of the information being transferred. The contents and format of the data signal is further defined by parameters.
error	1 – 256	Source □ Sink	A bit mask used to mark errors affecting the data being transferred in the current cycle. A single bit in error is used for each of the errors recognized by the component, as defined by the errorDescriptor property.
ready	1	Sink □ Source	Asserted high to indicate that the sink can accept data. ready is asserted by the sink on cycle <n> to mark cycle <n + readyLatency> as a ready cycle. The source may only assert valid and transfer data during ready cycles. Sources without a ready input cannot be backpressured. Sinks without a ready output never need to backpressure.
valid	1	Source □ Sink	Asserted by the source to qualify all other source to sink signals. The sink samples data and other source-to-sink signals on ready cycles where valid is asserted. All other cycles are ignored. Sources without a valid output implicitly provide valid data on every cycle that they are not being backpressured. Sinks without a valid input expect valid data on every cycle that they are not backpressuring.
<b>Packet Transfer Signals</b>			
empty	1 – 5	Source □ Sink	Indicates the number of symbols that are empty, that is, do not represent valid data. The empty signal is not used on interfaces where there is one symbol per beat.
endofpacket	1	Source □ Sink	Asserted by the source to mark the end of a packet.
startofpacket	1	Source □ Sink	Asserted by the source to mark the beginning of a packet.

### 11.14.9 Avalon Clock Source Signal Roles

An Avalon Clock source interface drives a clock signal out of a component.

**Table 180. Clock Source Signal Roles**

Signal Role	Width	Direction	Required	Description
clk	1	Output	Yes	An output clock signal.

### 11.14.10 Avalon Clock Sink Signal Roles

A clock sink provides a timing reference for other interfaces and internal logic.



**Table 181. Clock Sink Signal Roles**

Signal Role	Width	Direction	Required	Description
clk	1	Input	Yes	A clock signal. Provides synchronization for internal logic and for other interfaces.

### 11.14.11 Avalon Conduit Signal Roles

**Table 182. Conduit Signal Roles**

Signal Role	Width	Direction	Description
<any>	<n>	In, out, or bidirectional	A conduit interface consists of one or more input, output, or bidirectional signals of arbitrary width. Conduits can have any user-specified role. You can connect compatible Conduit interfaces inside a Platform Designer system provided the roles and widths match and the directions are opposite.

### 11.14.12 Avalon Tristate Conduit Signal Roles

The following table lists the signal defined for the Avalon Tristate Conduit interface. All Avalon-TC signals apply to both masters and slaves and have the same meaning for both

**Table 183. Tristate Conduit Interface Signal Roles**

Signal Role	Width	Direction	Required	Description
request	1	Master <input type="checkbox"/> Slave	Yes	The meaning of <code>request</code> depends on the state of the <code>grant</code> signal, as the following rules dictate. When <code>request</code> is asserted and <code>grant</code> is deasserted, <code>request</code> is requesting access for the current cycle. When <code>request</code> is asserted and <code>grant</code> is asserted, <code>request</code> is requesting access for the next cycle. Consequently, <code>request</code> should be deasserted on the final cycle of an access. The <code>request</code> is deasserted in the last cycle of a bus access. It can be reasserted immediately following the final cycle of a transfer. This protocol makes both re arbitration and continuous bus access possible if no other masters are requesting access. Once asserted, <code>request</code> must remain asserted until granted. Consequently, the shortest bus access is 2 cycles. Refer to <i>Tristate Conduit Arbitration Timing</i> for an example of arbitration timing.
grant	1	Slave <input type="checkbox"/> Master	Yes	When asserted, indicates that a tristate conduit master has been granted access to perform transactions. <code>grant</code> is asserted in response to the <code>request</code> signal. It remains asserted until 1 cycle following the deassertion of <code>request</code> .
<name>_in	1 - 1024	Slave <input type="checkbox"/> Master	No	The input signal of a logical tristate signal.
<name>_out	1 - 1024	Master <input type="checkbox"/> Slave	No	The output signal of a logical tristate signal.
<name>_outen	1	Master <input type="checkbox"/> Slave	No	The output enable for a logical tristate signal.



### 11.14.13 Avalon Tri-State Slave Interface Signal Types

**Table 184. Tri-state Slave Interface Signal Types**

Name	Width	Direction	Required	Description
address	1 - 32	input	No	Address lines to the slave port. Specifies a byte offset into the slave's address space.
read read_n	1	input	No	Read-request signal. Not required if the slave port never outputs data. If present, data must also be used.
write write_n	1	input	No	Write-request signal. Not required if the slave port never receives data from a master. If present, data must also be present, and writebyteenable cannot be present.
chipselect chipselect_n	1	input	No	When present, the slave port ignores all Avalon-MM signals unless chipselect is asserted. chipselect is always present in combination with read or write
outputenable outputenable_n	1	input	Yes	Output-enable signal. When deasserted, a tri-state slave port must not drive its data lines otherwise data contention may occur.
data	8,16, 32, 64, 128, 256, 512, 1024	bidir	No	Bidirectional data. During write transfers, the FPGA drives the data lines. During read transfers the slave device drives the data lines, and the FPGA captures the data signals and provides them to the master.
byteenable byteenable_n	2, 4, 8,16, 32, 64, 128	input	No	Enables specific byte lanes during transfers. Each bit in byteenable corresponds to a byte lane in data. During writes, byteenables specify which bytes the master is writing to the slave. During reads, byteenables indicates which bytes the master is reading. Slaves that simply return data with no side effects are free to ignore byteenables during reads. When more than one byte lane is asserted, all asserted lanes are guaranteed to be adjacent. The number of adjacent lines must be a power of 2, and the specified bytes must be aligned on an address boundary for the size of the data. The are legal values for a 32-bit slave:  <div style="border: 1px solid black; padding: 5px; width: fit-content;"> <pre> 1111 writes full 32 bits 0011 writes lower 2 bytes 1100 writes upper 2 bytes 0001 writes byte 0 only 0010 writes byte 1 only 0100 writes byte 2 only 1000 writes byte 3 only </pre> </div>

*continued...*



Name	Width	Direction	Required	Description
writebyteenable writebyteenable_n	2,4,8,16, 32, 64,128	input	No	Equivalent to the logical AND of the byteenable and write signals. When used, the write signal is not used.
begintransfer1	1	input	No	Asserted for the first cycle of each transfer.
<i>Note:</i> All Avalon signals are active high. Avalon signals that can also be asserted low list both versions in the <b>Signal Role</b> column.				

### 11.14.14 Avalon Interrupt Sender Signal Roles

**Table 185. Interrupt Sender Signal Roles**

Signal Role	Width	Direction	Required	Description
irq irq_n	1	Output	Yes	Interrupt Request. A slave asserts irq when it needs service. The interrupt receiver determines the relative priority of the interrupts.

### 11.14.15 Avalon Interrupt Receiver Signal Roles

**Table 186. Interrupt Receiver Signal Roles**

Signal Role	Width	Direction	Required	Description
irq	1-32	Input	Yes	irq is an <n>-bit vector, where each bit corresponds directly to one IRQ sender with no inherent assumption of priority.

## 11.15 Document Revision History

The table below indicates edits made to the *Platform Designer Interconnect* content since its creation.

**Table 187. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>Changed instances of <i>Qsys Pro</i> to Platform Designer</li> <li>Updated information about the Reset Sequencer.</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Implemented Intel rebranding.</li> <li>Implemented Platform Designer rebranding.</li> </ul>
2015.11.02	15.1.0	Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i> .
2015.05.04	15.0.0	<ul style="list-style-type: none"> <li>Fixed Priority Arbitration.</li> <li>Added topic: <i>Read and Write Responses</i>.</li> <li>Added topic: <i>Error Correction Coding (ECC) in Platform Designer Interconnect</i>.</li> <li>Added: response [1:0], <i>Avalon Memory-Mapped Interface Signal Roles</i>.</li> <li>Added writeresponsevalid, <i>Avalon Memory-Mapped Interface Signal Roles</i>.</li> </ul>
<b>continued...</b>		



Date	Version	Changes
December 2014	14.1.0	<ul style="list-style-type: none"><li>Read error responses, Avalon Memory-Mapped Interface Signal, response.</li><li>Burst Adapter Implementation Options: Generic converter (slower, lower area), Per-burst-type converter (faster, higher area).</li></ul>
August 2014	14.0a10.0	<ul style="list-style-type: none"><li>Updated Platform Designer Packet Format for Memory-Mapped Master and Slave Interfaces table, <i>Protection</i>.</li><li>Streaming Interface renamed to Avalon Streaming Interfaces.</li><li>Added <i>Response Merging</i> under <i>Memory-Mapped Interfaces</i>.</li></ul>
June 2014	14.0.0	<ul style="list-style-type: none"><li>AXI4-Lite support.</li><li>AXI4-Stream support.</li><li>Avalon-ST adapter parameters.</li><li>IRQ Bridge.</li><li>Handling Read Side Effects note added.</li></ul>
November 2013	13.1.0	<ul style="list-style-type: none"><li>HSSI clock support.</li><li>Reset Sequencer.</li><li>Interconnect pipelining.</li></ul>
May 2013	13.0.0	<ul style="list-style-type: none"><li>AMBA APB support.</li><li>Auto-inserted Avalon-ST adapters feature.</li><li>Moved Address Span Extender to the <i>Platform Designer System Design Components</i> chapter.</li></ul>
November 2012	12.1.0	<ul style="list-style-type: none"><li>AMBA AXI4 support.</li></ul>
June 2012	12.0.0	<ul style="list-style-type: none"><li>AMBA AXI3 support.</li><li>Avalon-ST adapters.</li><li>Address Span Extender.</li></ul>
November 2011	11.0.1	Template update.
May 2011	11.0.0	Removed beta status.
December 2010	10.1.0	Initial release.

### Related Links

#### [Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 12 Optimizing Platform Designer System Performance

---

You can optimize system interconnect performance for Intel designs that you create with the Platform Designer system integration tool.

*Note:* Intel now refers to Qsys Pro as Platform Designer.

The foundation of any system is the interconnect logic that connects hardware blocks or components. Creating interconnect logic is prone to errors, is time consuming to write, and is difficult to modify when design requirements change. The Platform Designer system integration tool addresses these issues and provides an automatically generated and optimized interconnect designed to satisfy your system requirements.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

*Note:* Recommended Intel practices may improve clock frequency, throughput, logic utilization, or power consumption of your Platform Designer design. When you design a Platform Designer system, use your knowledge of your design intent and goals to further optimize system performance beyond the automated optimization available in Platform Designer.

### Related Links

- [Creating a System with Platform Designer](#) on page 327
- [Creating Platform Designer Components](#) on page 608
- [Platform Designer Interconnect](#) on page 659
- [Avalon Interface Specifications](#)
- [AMBA Protocol Specifications](#)

### 12.1 Designing with Avalon and AXI Interfaces

Platform Designer Avalon and AXI interconnect for memory-mapped interfaces is flexible, partial crossbar logic that connects master and slave interfaces.

Avalon Streaming (Avalon-ST) links connect point-to-point, unidirectional interfaces and are typically used in data stream applications. Each pair of components is connected without any requirement to arbitrate between the data source and sink.

Because Platform Designer supports multiplexed memory-mapped and streaming connections, you can implement systems that use multiplexed logic for control and streaming for data in a single design.

### Related Links

[Creating Platform Designer Components](#) on page 608

### 12.1.1 Designing Streaming Components

When you design streaming component interfaces, you must consider integration and communication for each component in the system. One common consideration is buffering data internally to accommodate latency between components.

For example, if the component's Avalon-ST output or source of streaming data is back-pressured because the ready signal is deasserted, then the component must back-pressure its input or sink interface to avoid overflow.

You can use a FIFO to back-pressure internally on the output side of the component so that the input can accept more data even if the output is back-pressured. Then, you can use the FIFO almost full flag to back-pressure the sink interface or input data when the FIFO has only enough space to satisfy the internal latency. You can drive the data valid signal of the output or source interface with the FIFO not empty flag when that data is available.

### 12.1.2 Designing Memory-Mapped Components

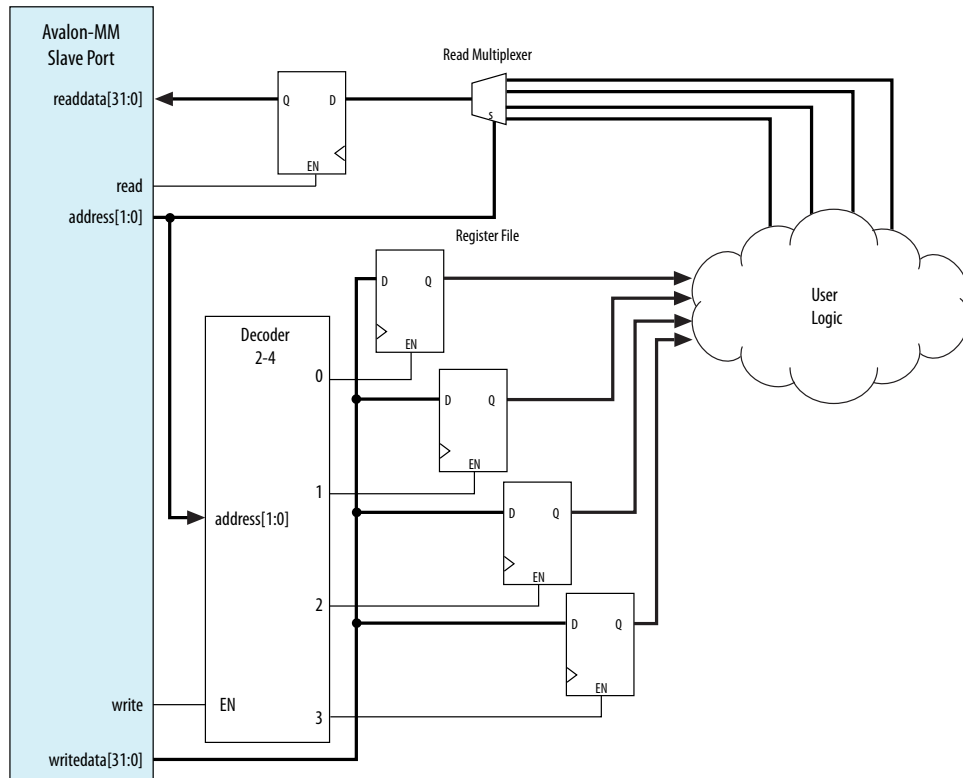
When designing with memory-mapped components, you can implement any component that contains multiple registers mapped to memory locations, for example, a set of four output registers to support software read back from logic. Components that implement read and write memory-mapped transactions require three main building blocks: an address decoder, a register file, and a read multiplexer.

The decoder enables the appropriate 32-bit or 64-bit register for writes. For reads, the address bits drive the multiplexer selection bits. The read signal registers the data from the multiplexer, adding a pipeline stage so that the component can achieve a higher clock frequency.





Figure 243. Control and Status Registers (CSR) in a Slave Component



This slave component has four write wait states and one read wait state. Alternatively, if you want high throughput, you may set both the read and write wait states to zero, and then specify a read latency of one, because the component also supports pipelined reads.

## 12.2 Using Hierarchy in Systems

You can use hierarchy to sub-divide a system into smaller subsystems that you can then connect in a top-level Platform Designer system. Additionally, if a design contains one or more identical functional units, the functional unit can be defined as a subsystem and instantiated multiple times within a top-level system.

Hierarchy can simplify verification control of slaves connected to each master in a memory-mapped system. Before you implement subsystems in your design, you should plan the system hierarchical blocks at the top-level, using the following guidelines:

- **Plan shared resources**—Determine the best location for shared resources in the system hierarchy. For example, if two subsystems share resources, add the components that use those resources to a higher-level system for easy access.
- **Plan shared address space between subsystems**—Planning the address space ensures you can set appropriate sizes for bridges between subsystems.
- **Plan how much latency you may need to add to your system**—When you add an Avalon-MM Pipeline Bridge between subsystems, you may add latency to the overall system. You can reduce the added latency by parameterizing the bridge with zero cycles of latency, and by turning off the pipeline command and response signals.

Figure 244. Avalon-MM Pipeline Bridge

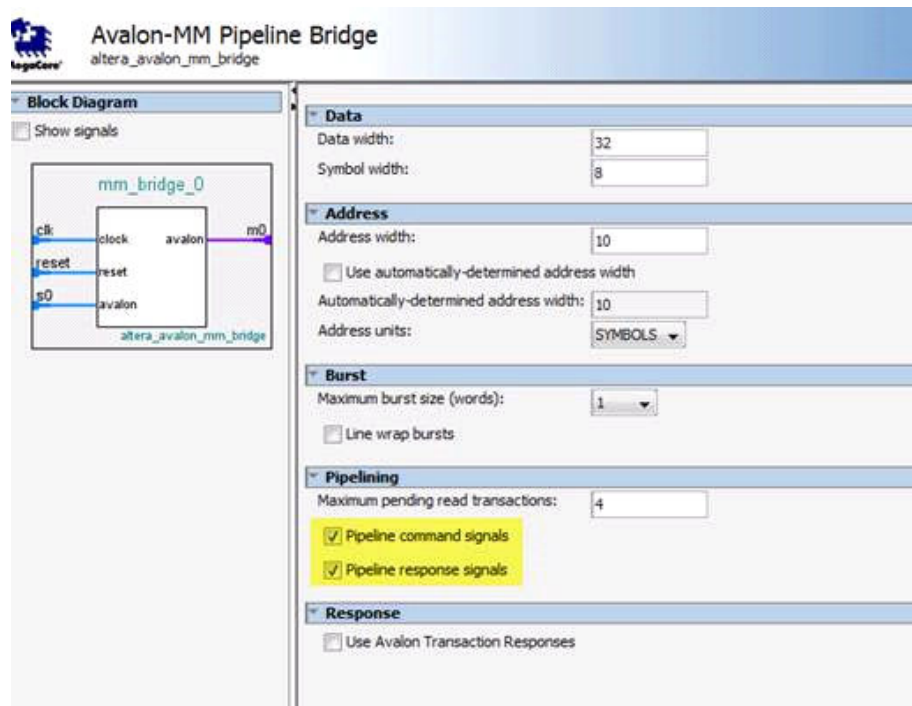
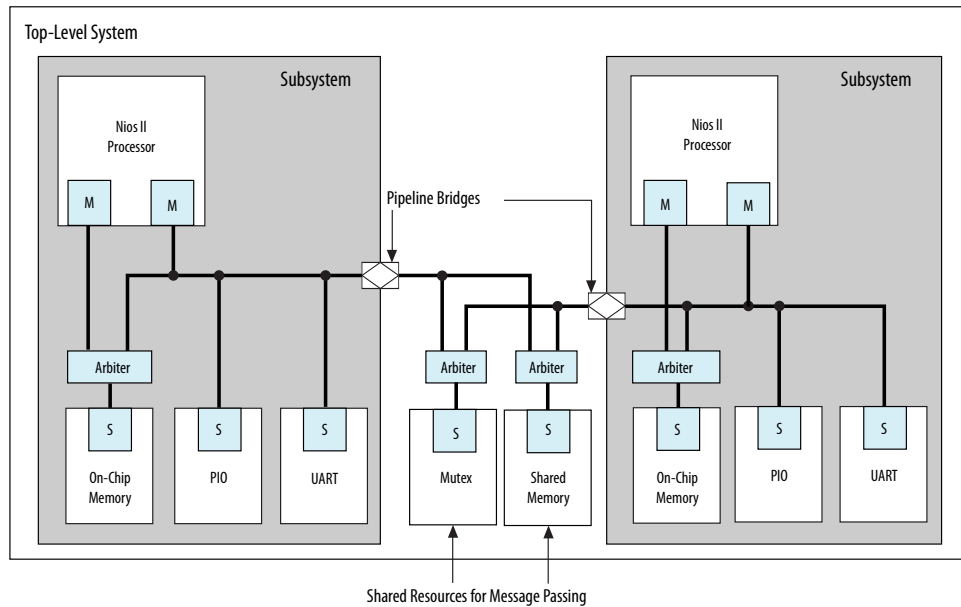


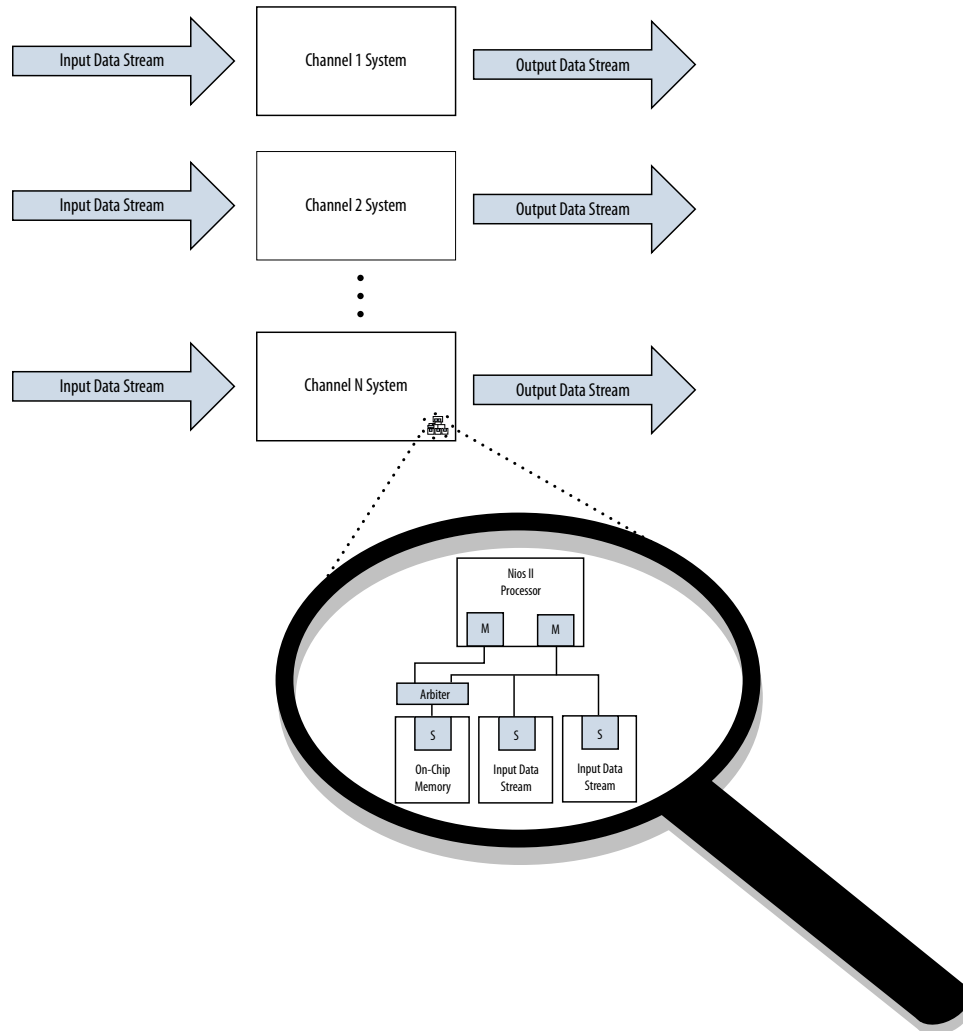


Figure 245. Passing Messages Between Subsystems



In this example, two Nios II processor subsystems share resources for message passing. Bridges in each subsystem export the Nios II data master to the top-level system that includes the mutex (mutual exclusion component) and shared memory component (which could be another on-chip RAM, or a controller for an off-chip RAM device).

**Figure 246. Multi Channel System**



You can also design systems that process multiple data channels by instantiating the same subsystem for each channel. This approach is easier to maintain than a larger, non-hierarchical system. Additionally, such systems are easier to scale because you can calculate the required resources as a multiple of the subsystem requirements.

**Related Links**

[Avalon-MM Pipeline Bridge](#)

**12.3 Using Concurrency in Memory-Mapped Systems**

Platform Designer interconnect uses parallel hardware in FPGAs, which allows you to design concurrency into your system and process transactions simultaneously.



### 12.3.1 Implementing Concurrency With Multiple Masters

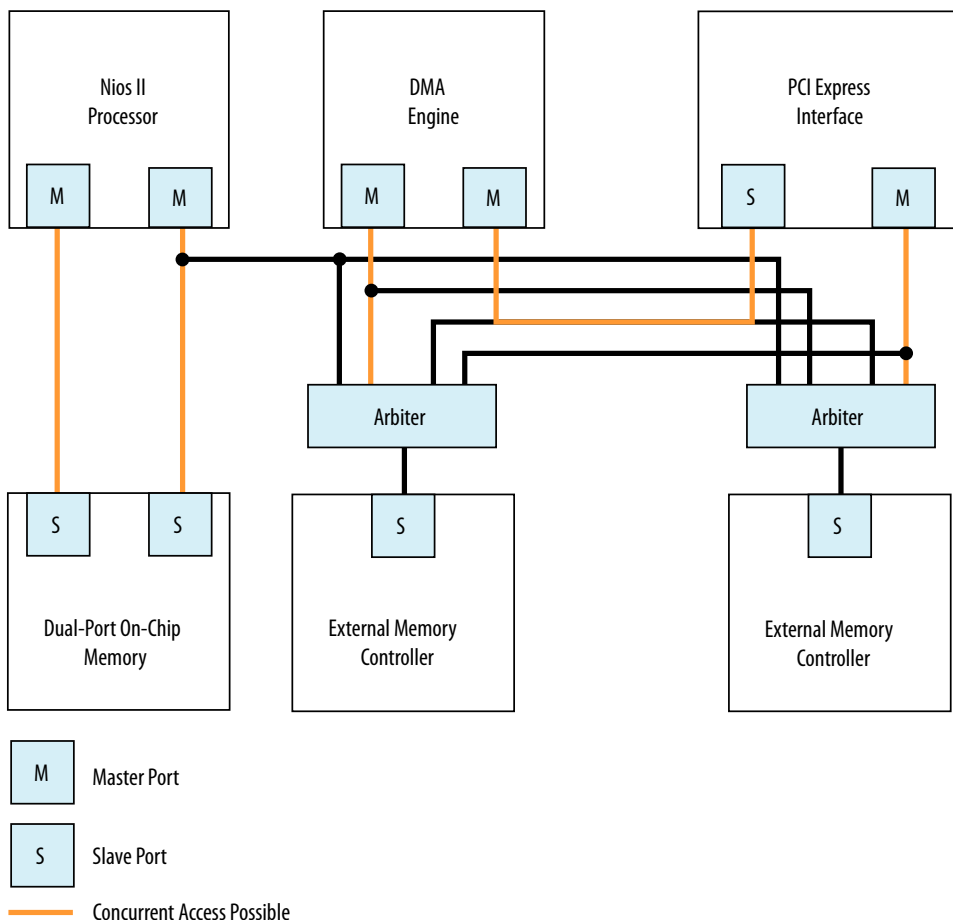
Implementing concurrency requires multiple masters in a Platform Designer system. Systems that include a processor contain at least two master interfaces because the processors include separate instruction and data masters. You can categorize master components as follows:

- General purpose processors, such as Nios II processors
- DMA (direct memory access) engines
- Communication interfaces, such as PCI Express

Because Platform Designer generates an interconnect with slave-side arbitration, every master interface in a system can issue transfers concurrently, if they are not posting transfers to the same slave. Concurrency is limited by the number of master interfaces sharing any particular slave interface. If a design requires higher data throughput, you can increase the number of master and slave interfaces to increase the number of transfers that occur simultaneously. The example below shows a system with three master interfaces.

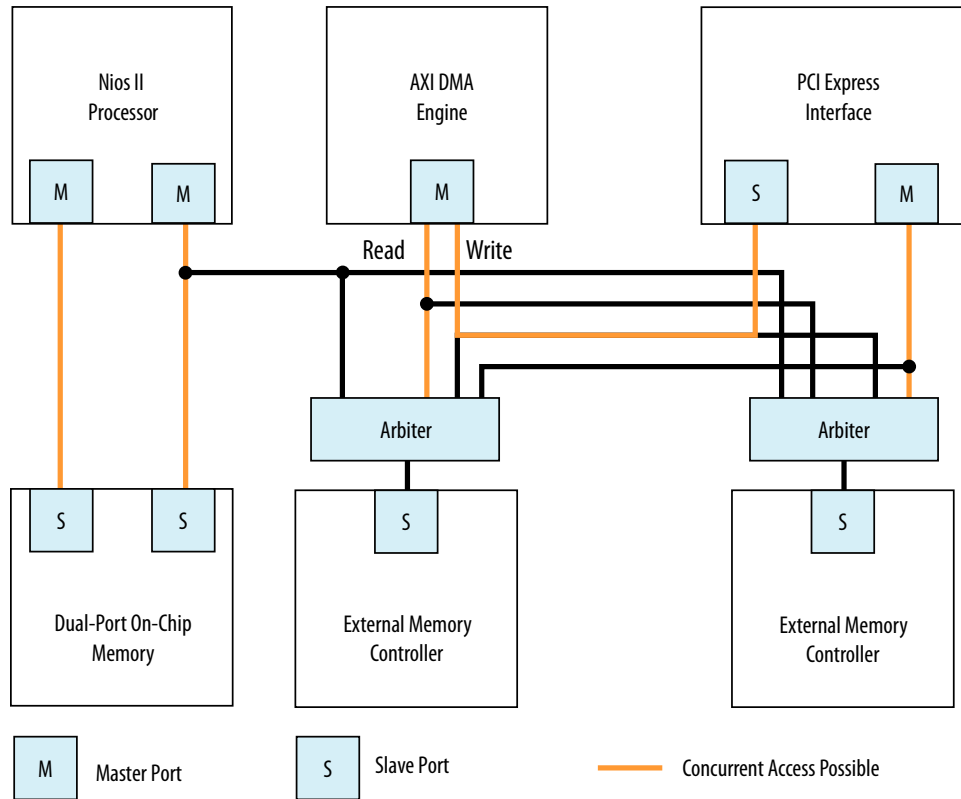
**Figure 247. Avalon Multiple Master Parallel Access**

In this Avalon example, the DMA engine operates with Avalon-MM read and write masters. The yellow lines represent active simultaneous connections.



**Figure 248. AXI Multiple Master Parallel Access**

In this example, the DMA engine operates with a single master, because in AXI, the write and read channels on the master are independent and can process transactions simultaneously. There is concurrency between the read and write channels, with the yellow lines representing concurrent datapaths.

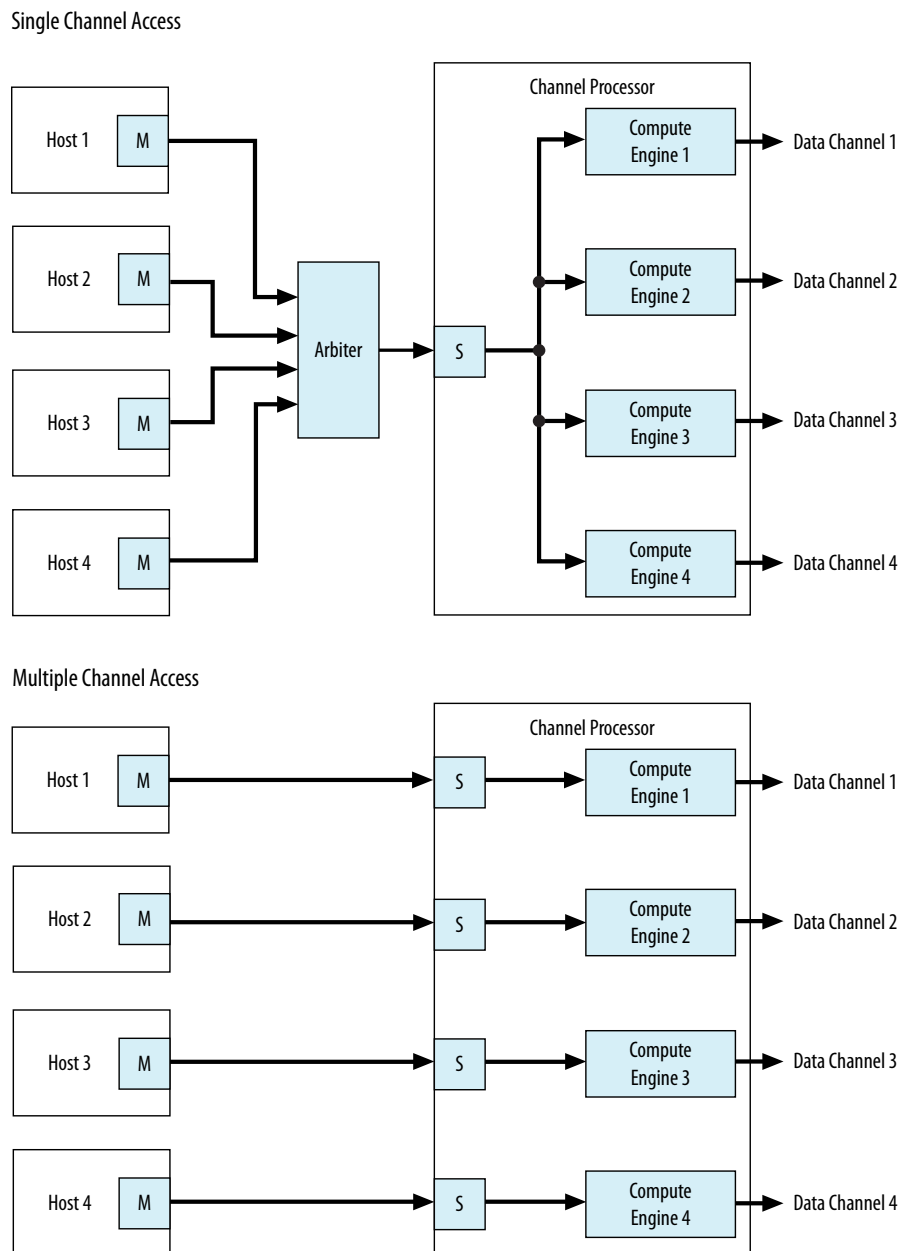


### 12.3.2 Implementing Concurrency With Multiple Slaves

You can create multiple slave interfaces for a particular function to increase concurrency in your design.



Figure 249. Single Interface Versus Multiple Interfaces



In this example, there are two channel processing systems. In the first, four hosts must arbitrate for the single slave interface of the channel processor. In the second, each host drives a dedicated slave interface, allowing all master interfaces to simultaneously access the slave interfaces of the component. Arbitration is not necessary when there is a single host and slave interface.

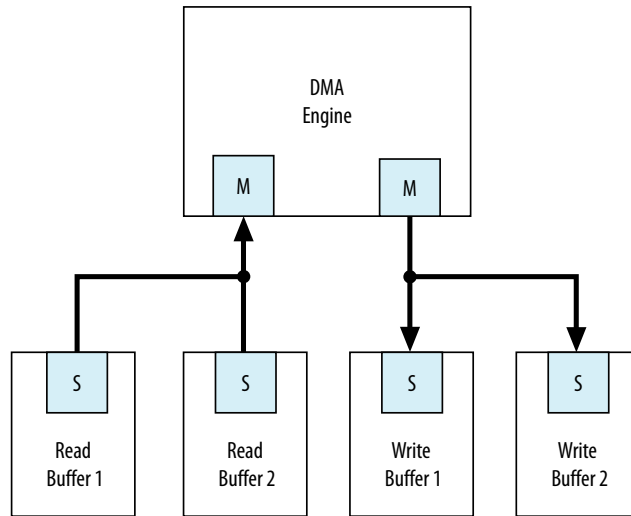
### 12.3.3 Implementing Concurrency with DMA Engines

In some systems, you can use DMA engines to increase throughput. You can use a DMA engine to transfer blocks of data between interfaces, which then frees the CPU from doing this task. A DMA engine transfers data between a programmed start and end address without intervention, and the data throughput is dictated by the components connected to the DMA. Factors that affect data throughput include data width and clock frequency.

**Figure 250. Single or Dual DMA Channels**

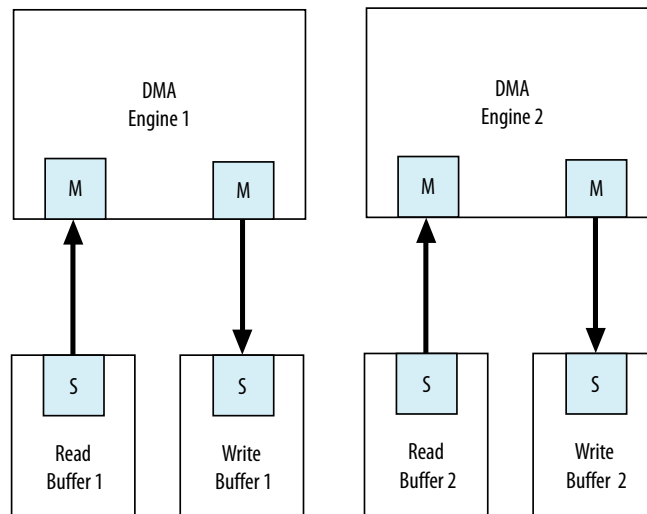
#### Single DMA Channel

Maximum of One Read & One Write Per Clock Cycle



#### Dual DMA Channels

Maximum of Two Reads & Two Writes Per Clock Cycle







In this example, the system can sustain more concurrent read and write operations by including more DMA engines. Accesses to the read and write buffers in the top system are split between two DMA engines, as shown in the Dual DMA Channels at the bottom of the figure.

The DMA engine operates with Avalon-MM write and read masters. An AXI DMA typically has only one master, because in AXI, the write and read channels on the master are independent and can process transactions simultaneously.

## 12.4 Inserting Pipeline Stages to Increase System Frequency

Platform Designer provides the **Limit interconnect pipeline stages to** option on the **Interconnect Requirements** tab to automatically add pipeline stages to the Platform Designer interconnect when you generate a system.

You can specify between 0 to 4 pipeline stages, where 0 means that the interconnect has a combinational datapath. You can specify a unique interconnect pipeline stage value for each subsystem.

Adding pipeline stages may increase the  $f_{MAX}$  of the design by reducing the combinational logic depth, at the cost of additional latency and logic utilization.

The insertion of pipeline stages requires certain interconnect components. For example, in a system with a single slave interface, there is no multiplexer; therefore multiplexer pipelining does not occur. When there is an Avalon or AXI single-master to single-slave system, no pipelining occurs, regardless of the **Limit interconnect pipeline stages to** option.

### Related Links

- [Interconnect Pipelining](#) on page 715
- [Pipelined Avalon-MM Interfaces](#) on page 769
- [Creating a System with Platform Designer](#) on page 327

## 12.5 Using Bridges

You can use bridges to increase system frequency, minimize generated Platform Designer logic, minimize adapter logic, and to structure system topology when you want to control where Platform Designer adds pipelining. You can also use bridges with arbiters when there is concurrency in the system.

An Avalon bridge has an Avalon-MM slave interface and an Avalon-MM master interface. You can have many components connected to the bridge slave interface, or many components connected to the bridge master interface. You can also have a single component connected to a single bridge slave or master interface.

You can configure the data width of the bridge, which can affect how Platform Designer generates bus sizing logic in the interconnect. Both interfaces support Avalon-MM pipelined transfers with variable latency, and can also support configurable burst lengths.

Transfers to the bridge slave interface are propagated to the master interface, which connects to components downstream from the bridge. When you need greater control over interconnect pipelining, you can use bridges instead of the **Limit Interconnect Pipeline Stages to** option.

**Note:** You can use Avalon bridges between AXI interfaces, and between Avalon domains. Platform Designer automatically creates interconnect logic between the AXI and Avalon interfaces, so you do not have to explicitly instantiate bridges between these domains. For more discussion about the benefits and disadvantages of shared and separate domains, refer to the *Platform Designer Interconnect*.

**Related Links**

- [Bridges](#) on page 914
- [AMBA 3 APB Protocol Specification Support \(version 1.0\)](#) on page 722

**12.5.1 Using Bridges to Increase System Frequency**

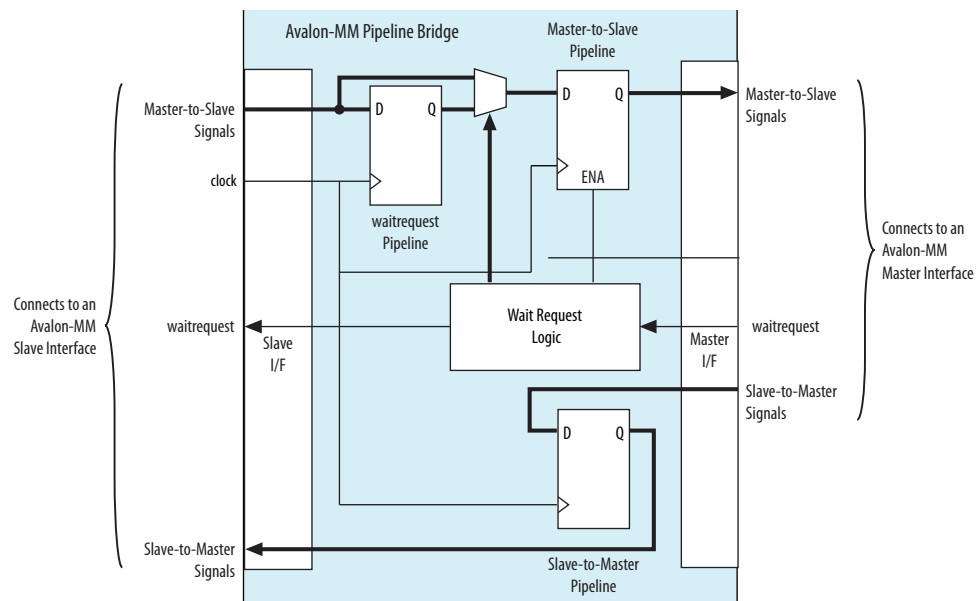
In Platform Designer, you can introduce interconnect pipeline stages or pipeline bridges to increase clock frequency in your system. Bridges control the system interconnect topology and allow you to subdivide the interconnect, giving you more control over pipelining and clock crossing functionality.

**12.5.1.1 Inserting Pipeline Bridges**

You can insert an Avalon-MM pipeline bridge to insert registers in the path between the bridges and its master and slaves. If a critical register-to-register delay occurs in the interconnect, a pipeline bridge can help reduce this delay and improve system  $f_{MAX}$ .

The Avalon-MM pipeline bridge component integrates into any Platform Designer system. The pipeline bridge options can increase logic utilization and read latency. The change in topology may also reduce concurrency if multiple masters arbitrate for the bridge. You can use the Avalon-MM pipeline bridge to control topology without adding a pipeline stage. A pipeline bridge that does not add a pipeline stage is optimal in some latency-sensitive applications. For example, a CPU may benefit from minimal latency when accessing memory.

**Figure 251. Avalon-MM Pipeline Bridge**



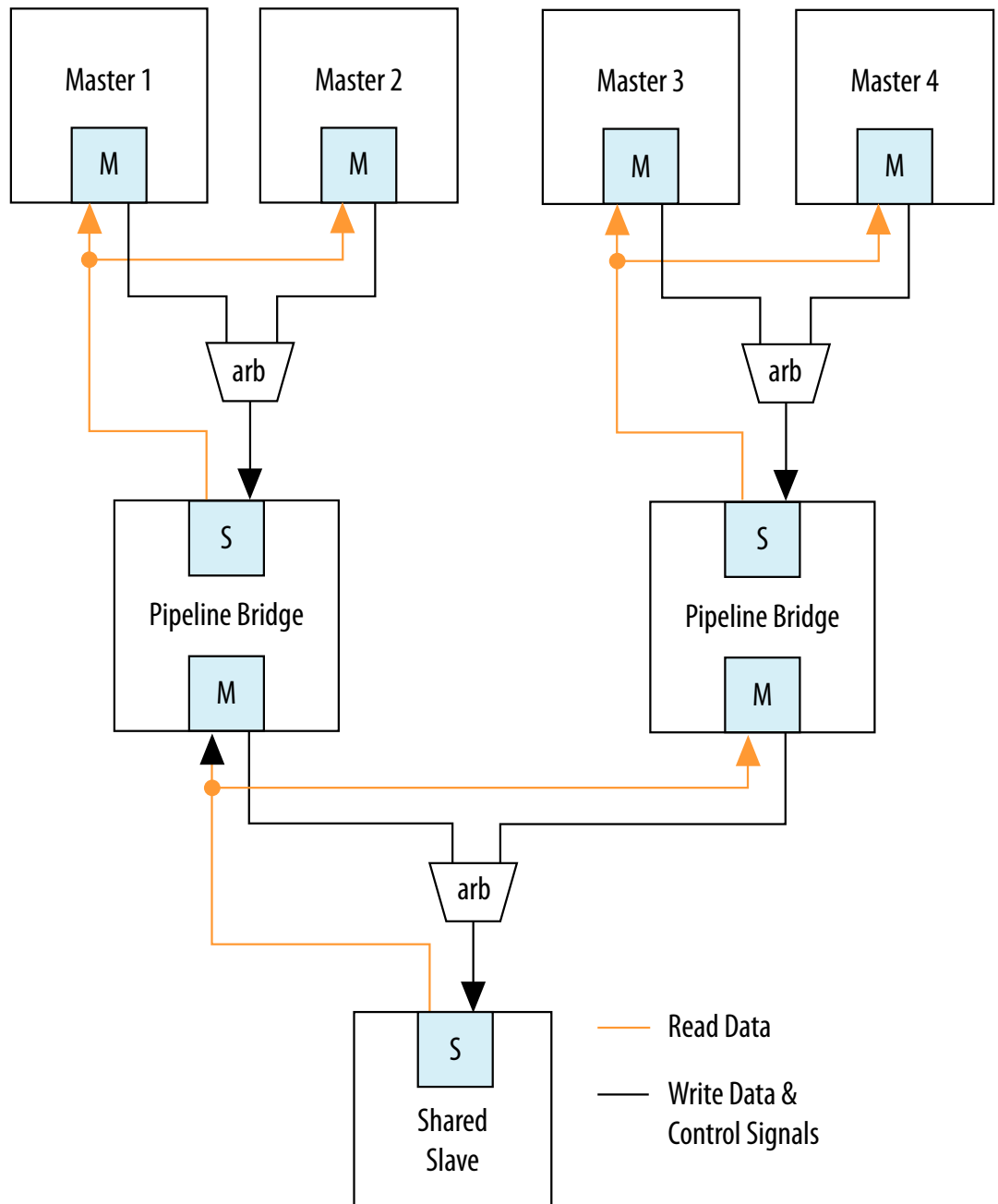


#### **12.5.1.1.1 Implementing Command Pipelining (Master-to-Slave)**

When multiple masters share a slave device, you can use command pipelining to improve performance.

The arbitration logic for the slave interface must multiplex the `address`, `writedata`, and `burstcount` signals. The multiplexer width increases proportionally with the number of masters connecting to a single slave interface. The increased multiplexer width may become a timing critical path in the system. If a single pipeline bridge does not provide enough pipelining, you can instantiate multiple instances of the bridge in a tree structure to increase the pipelining and further reduce the width of the multiplexer at the slave interface.

Figure 252. Tree of Bridges



### 12.5.1.1.2 Implementing Response Pipelining (Slave-to-Master)

When masters connect to multiple slaves that support read transfers, you can use slave-to-master pipelining to improve performance.



The interconnect inserts a multiplexer for every read datapath back to the master. As the number of slaves supporting read transfers connecting to the master increases, the width of the read data multiplexer also increases. If the performance increase is insufficient with one bridge, you can use multiple bridges in a tree structure to improve  $f_{MAX}$ .

### 12.5.1.2 Using Clock Crossing Bridges

The clock crossing bridge contains a pair of clock crossing FIFOs, which isolate the master and slave interfaces in separate, asynchronous clock domains. Transfers to the slave interface are propagated to the master interface.

When you use a FIFO clock crossing bridge for the clock domain crossing, you add data buffering. Buffering allows pipelined read masters to post multiple reads to the bridge, even if the slaves downstream from the bridge do not support pipelined transfers.

You can also use a clock crossing bridge to place high and low frequency components in separate clock domains. If you limit the fast clock domain to the portion of your design that requires high performance, you may achieve a higher  $f_{MAX}$  for this portion of the design. For example, the majority of processor peripherals in embedded designs do not need to operate at high frequencies, therefore, you do not need to use a high-frequency clock for these components. When you compile a design with the Intel Quartus Prime software, compilation may take more time when the clock frequency requirements are difficult to meet because the Fitter needs more time to place registers to achieve the required  $f_{MAX}$ . To reduce the amount of effort that the Fitter uses on low priority and low performance components, you can place these behind a clock crossing bridge operating at a lower frequency, allowing the Fitter to increase the effort placed on the higher priority and higher frequency datapaths.

### 12.5.2 Using Bridges to Minimize Design Logic

Bridges can reduce interconnect logic by reducing the amount of arbitration and multiplexer logic that Platform Designer generates. This reduction occurs because bridges limit the number of concurrent transfers that can occur.

#### 12.5.2.1 Avoiding Speed Optimizations That Increase Logic

You can add an additional pipeline stage with a pipeline bridge between masters and slaves to reduce the amount of combinational logic between registers, which can increase system performance. If you can increase the  $f_{MAX}$  of your design logic, you may be able to turn off the Intel Quartus Prime software optimization settings, such as the **Perform register duplication** setting. Register duplication creates duplicate registers in two or more physical locations in the FPGA to reduce register-to-register delays. You may also want to choose **Speed** for the optimization method, which typically results in higher logic utilization due to logic duplication. By making use of the registers or FIFOs available in the bridges, you can increase the design speed and avoid needless logic duplication or speed optimizations, thereby reducing the logic utilization of the design.



### 12.5.2.2 Limiting Concurrency

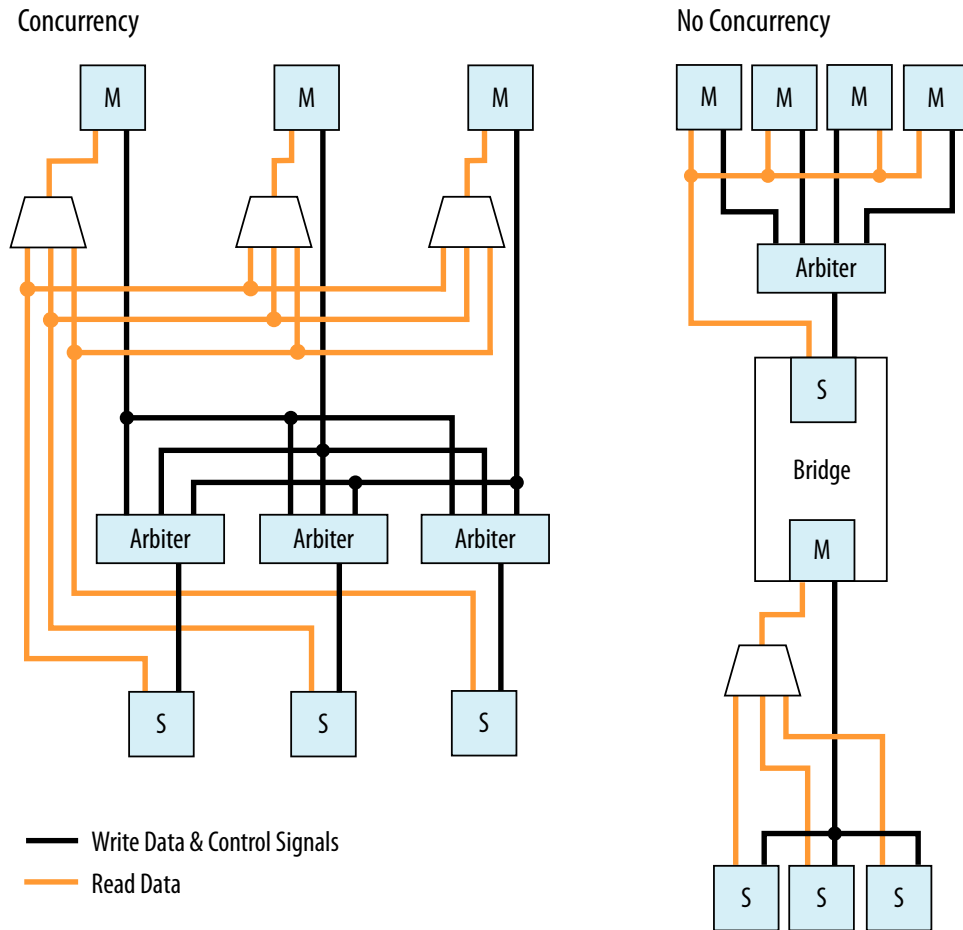
The amount of logic generated for the interconnect often increases as the system becomes larger because Platform Designer creates arbitration logic for every slave interface that is shared by multiple master interfaces. Platform Designer inserts multiplexer logic between master interfaces that connect to multiple slave interfaces if both support read datapaths.

Most embedded processor designs contain components that are either incapable of supporting high data throughput, or do not need to be accessed frequently. These components can contain master or slave interfaces. Because the interconnect supports concurrent accesses, you may want to limit concurrency by inserting bridges into the datapath to limit the amount of arbitration and multiplexer logic generated.

For example, if a system contains three master and three slave interfaces that are interconnected, Platform Designer generates three arbiters and three multiplexers for the read datapath. If these masters do not require a significant amount of simultaneous throughput, you can reduce the resources that your design consumes by connecting the three masters to a pipeline bridge. The bridge controls the three slave interfaces and reduces the interconnect into a bus structure. Platform Designer creates one arbitration block between the bridge and the three masters, and a single read datapath multiplexer between the bridge and three slaves, and prevents concurrency. This implementation is similar to a standard bus architecture.

You should not use this method for high throughput datapaths to ensure that you do not limit overall system performance.

**Figure 253. Differences Between Systems With and Without a Pipeline Bridge**



### 12.5.3 Using Bridges to Minimize Adapter Logic

Platform Designer generates adapter logic for clock crossing, width adaptation, and burst support when there is a mismatch between the clock domains, widths, or bursting capabilities of the master and slave interface pairs.

Platform Designer creates burst adapters when the maximum burst length of the master is greater than the master burst length of the slave. The adapter logic creates extra logic resources, which can be substantial when your system contains master interfaces connected to many components that do not share the same characteristics. By placing bridges in your design, you can reduce the amount of adapter logic that Platform Designer generates.

#### 12.5.3.1 Determining Effective Placement of Bridges

To determine the effective placement of a bridge, you should initially analyze each master in your system to determine if the connected slave devices support different bursting capabilities or operate in a different clock domain. The maximum burstcount of a component is visible as the `burstcount` signal in the HDL file of the component.

The maximum burst length is  $2^{(\text{width}(\text{burstcount} - 1))}$ , therefore, if the `burstcount` width is four bits, the maximum burst length is eight. If no `burstcount` signal is present, the component does not support bursting or has a burst length of 1.

To determine if the system requires a clock crossing adapter between the master and slave interfaces, check the **Clock** column for the master and slave interfaces. If the clock is different for the master and slave interfaces, Platform Designer inserts a clock crossing adapter between them. To avoid creating multiple adapters, you can place the components containing slave interfaces behind a bridge so that Platform Designer creates a single adapter. By placing multiple components with the same burst or clock characteristics behind a bridge, you limit concurrency and the number of adapters.

You can also use a bridge to separate AXI and Avalon domains to minimize burst adaptation logic. For example, if there are multiple Avalon slaves that are connected to an AXI master, you can consider inserting a bridge to access the adaptation logic once before the bridge, instead of once per slave. This implementation results in latency, and you would also lose concurrency between reads and writes.

### 12.5.3.2 Changing the Response Buffer Depth

When you use automatic clock-crossing adapters, Platform Designer determines the required depth of FIFO buffering based on the slave properties. If a slave has a high **Maximum Pending Reads** parameter, the resulting deep response buffer FIFO that Platform Designer inserts between the master and slave can consume a lot of device resources. To control the response FIFO depth, you can use a clock crossing bridge and manually adjust its FIFO depth to trade off throughput with smaller memory utilization.

For example, if you have masters that cannot saturate the slave, you do not need response buffering. Using a bridge reduces the FIFO memory depth and reduces the **Maximum Pending Reads** available from the slave.

### 12.5.4 Considering the Effects of Using Bridges

Before you use pipeline or clock crossing bridges in a design, you should carefully consider their effects. Bridges can have any combination of consequences on your design, which could be positive or negative. Benchmarking your system before and after inserting bridges can help you determine the impact to the design.

#### 12.5.4.1 Increased Latency

Adding a bridge to a design has an effect on the read latency between the master and the slave. Depending on the system requirements and the type of master and slave, this latency increase may not be acceptable in your design.

##### 12.5.4.1.1 Acceptable Latency Increase

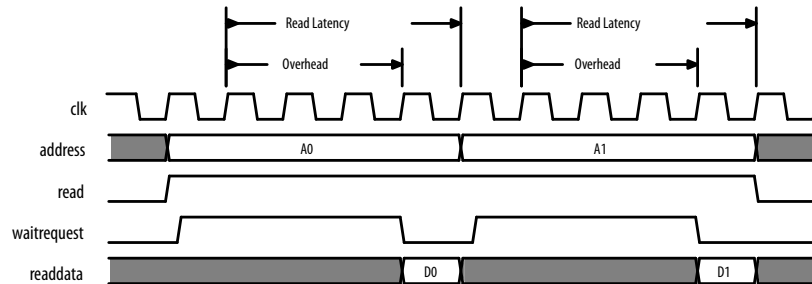
For a pipeline bridge, Platform Designer adds a cycle of latency for each pipeline option that is enabled. The buffering in the clock crossing bridge also adds latency. If you use a pipelined or burst master that posts many read transfers, the increase in latency does not impact performance significantly because the latency increase is very small compared to the length of the data transfer.





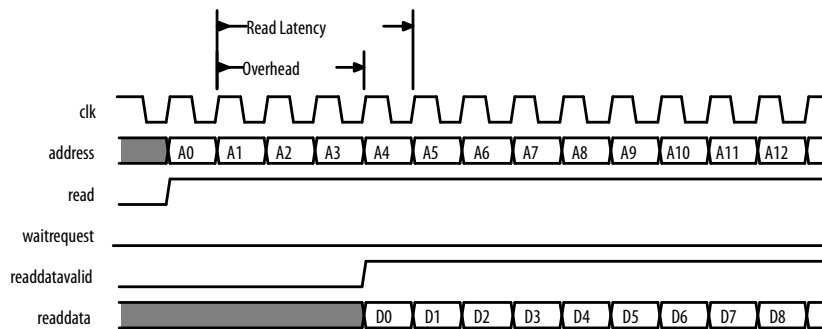
For example, if you use a pipelined read master such as a DMA controller to read data from a component with a fixed read latency of four clock cycles, but only perform a single word transfer, the overhead is three clock cycles out of the total of four. This is true when there is no additional pipeline latency in the interconnect. The read throughput is only 25%.

**Figure 254. Low-Efficiency Read Transfer**



However, if 100 words of data are transferred without interruptions, the overhead is three cycles out of the total of 103 clock cycles. This corresponds to a read efficiency of approximately 97% when there is no additional pipeline latency in the interconnect. Adding a pipeline bridge to this read path adds two extra clock cycles of latency. The transfer requires 105 cycles to complete, corresponding to an efficiency of approximately 94%. Although the efficiency decreased by 3%, adding the bridge may increase the  $f_{MAX}$  by 5%. For example, if the clock frequency can be increased, the overall throughput would improve. As the number of words transferred increases, the efficiency increases to nearly 100%, whether or not a pipeline bridge is present.

**Figure 255. High Efficiency Read Transfer**

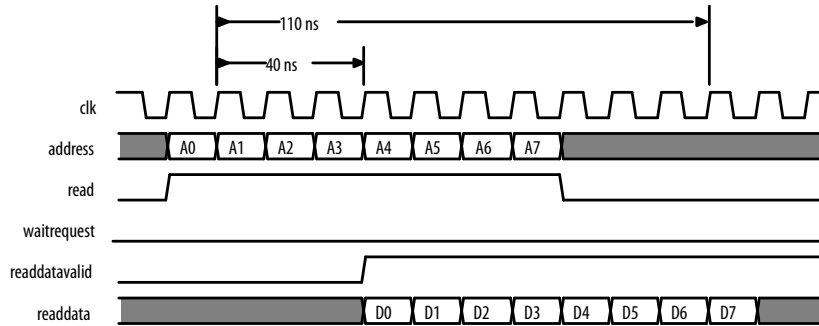


### 12.5.4.1.2 Unacceptable Latency Increase

Processors are sensitive to high latency read times and typically retrieve data for use in calculations that cannot proceed until the data arrives. Before adding a bridge to the datapath of a processor instruction or data master, determine whether the clock frequency increase justifies the added latency.

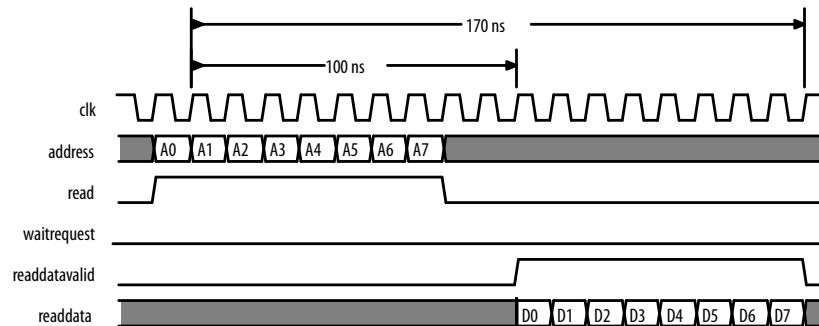
A Nios II processor instruction master has a cache memory with a read latency of four cycles, which is eight sequential words of data return for each read. At 100 MHz, the first read takes 40 ns to complete. Each successive word takes 10 ns so that eight reads complete in 110 ns.

**Figure 256. Performance of a Nios II Processor and Memory Operating at 100 MHz**



Adding a clock crossing bridge allows the memory to operate at 125 MHz. However, this increase in frequency is negated by the increase in latency because if the clock crossing bridge adds six clock cycles of latency at 100 MHz, then the memory continues to operate with a read latency of four clock cycles. Consequently, the first read from memory takes 100 ns, and each successive word takes 10 ns because reads arrive at the frequency of the processor, which is 100 MHz. In total, eight reads complete after 170 ns. Although the memory operates at a higher clock frequency, the frequency at which the master operates limits the throughput.

**Figure 257. Performance of a Nios II Processor and Eight Reads with Ten Cycles Latency**

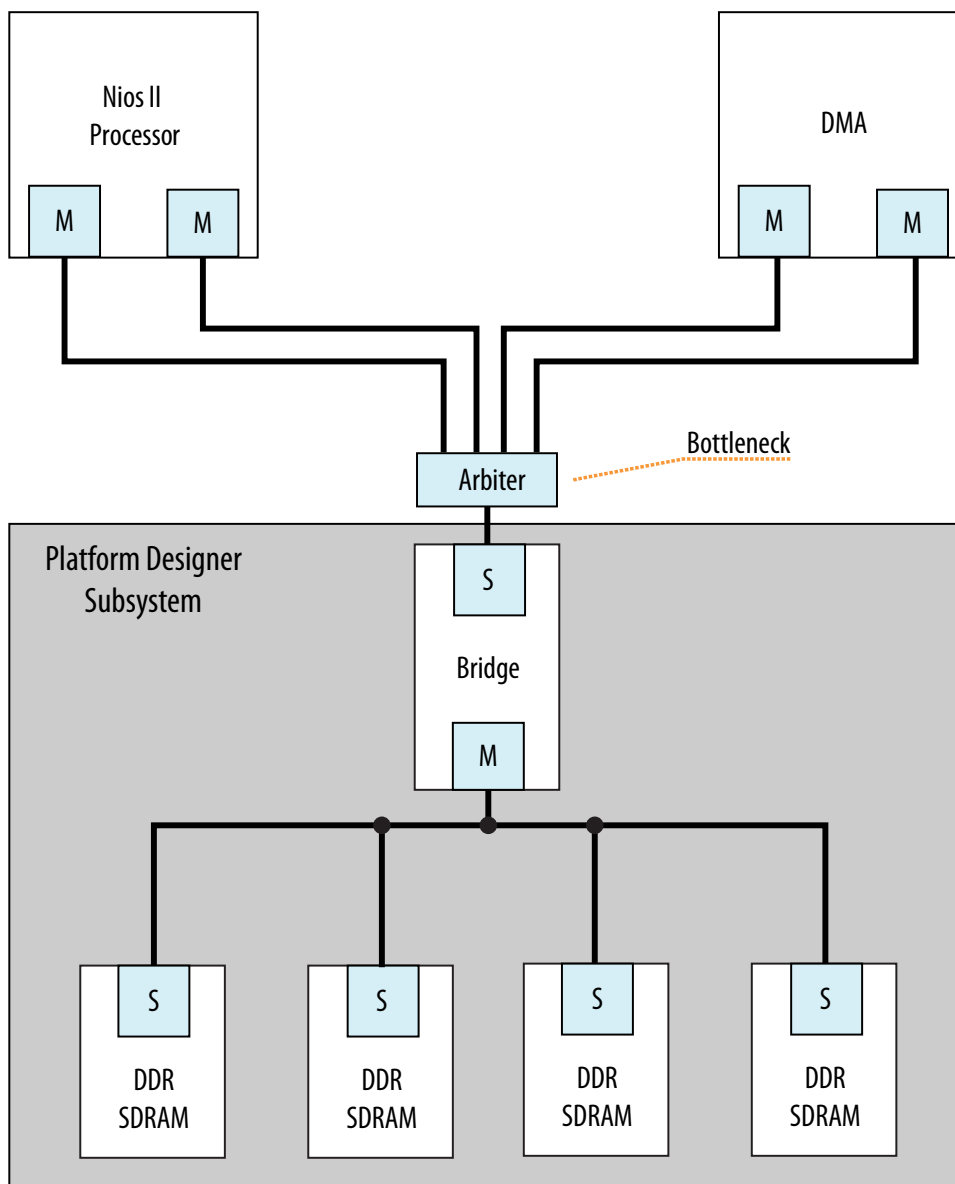


### 12.5.4.2 Limited Concurrency

Placing a bridge between multiple master and slave interfaces limits the number of concurrent transfers your system can initiate. This limitation is the same when connecting multiple master interfaces to a single slave interface. The slave interface of the bridge is shared by all the masters and, as a result, Platform Designer creates arbitration logic. If the components placed behind a bridge are infrequently accessed, this concurrency limitation may be acceptable.

Bridges can have a negative impact on system performance if you use them inappropriately. For example, if multiple memories are used by several masters, you should not place the memory components behind a bridge. The bridge limits memory performance by preventing concurrent memory accesses. Placing multiple memory components behind a bridge can cause the separate slave interfaces to appear as one large memory to the masters accessing the bridge; all masters must access the same slave interface.

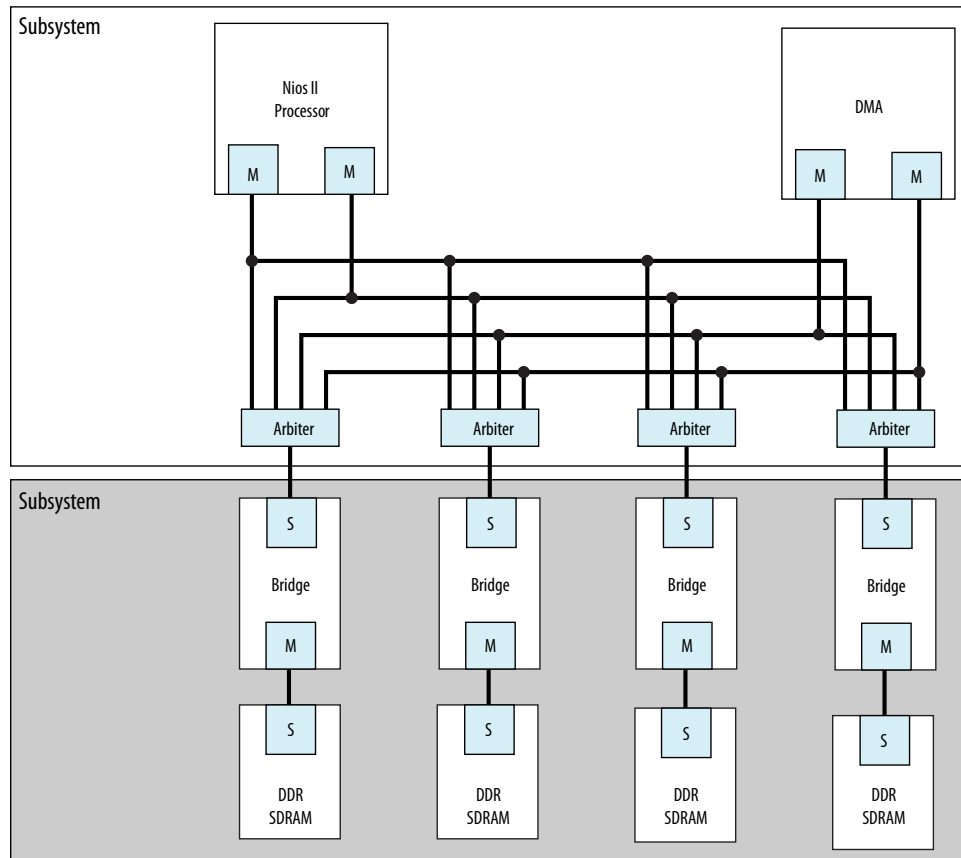
Figure 258. Inappropriate Use of a Bridge in a Hierarchical System



A memory subsystem with one bridge that acts as a single slave interface for the Avalon-MM Nios II and DMA masters, which results in a bottleneck architecture. The bridge acts as a bottleneck between the two masters and the memories.

If the  $f_{MAX}$  of your memory interfaces is low and you want to use a pipeline bridge between subsystems, you can place each memory behind its own bridge, which increases the  $f_{MAX}$  of the system without sacrificing concurrency.

Figure 259. Efficient Memory Pipelining Without a Bottleneck in a Hierarchical System



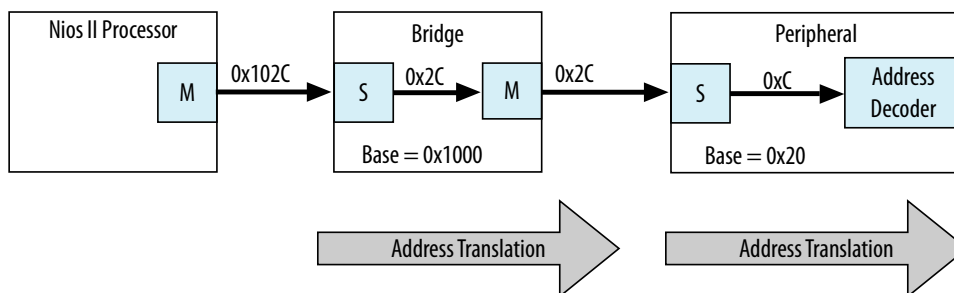
### 12.5.4.3 Address Space Translation

The slave interface of a pipeline or clock crossing bridge has a base address and address span. You can set the base address, or allow Platform Designer to set it automatically. The address of the slave interface is the base offset address of all the components connected to the bridge. The address of components connected to the bridge is the sum of the base offset and the address of that component.

The master interface of the bridge drives only the address bits that represent the offset from the base address of the bridge slave interface. Any time a master accesses a slave through a bridge, both addresses must be added together, otherwise the transfer fails. The **Address Map** tab displays the addresses of the slaves connected to each master and includes address translations caused by system bridges.



**Figure 260. Bridge Address Translation**

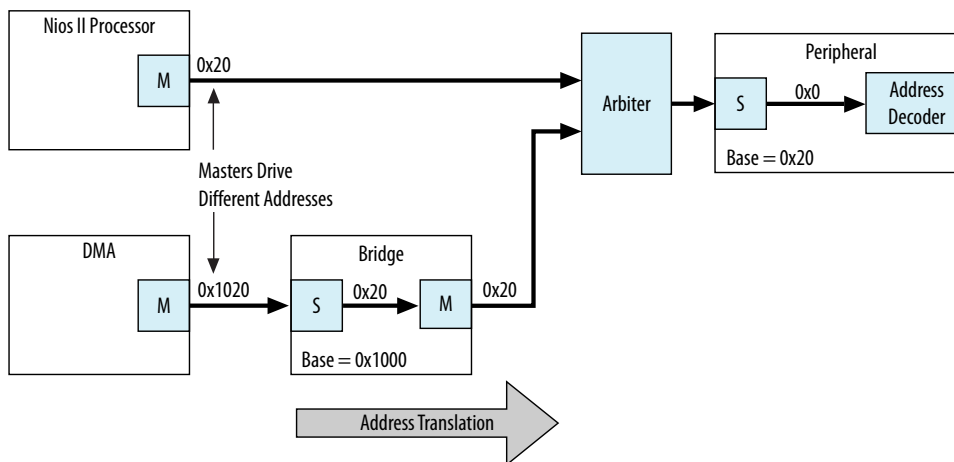


In this example, the Nios II processor connects to a bridge located at base address 0x1000, a slave connects to the bridge master interface at an offset of 0x20, and the processor performs a write transfer to the fourth 32-bit or 64-bit word within the slave. Nios II drives the address 0x102C to interconnect, which is within the address range of the bridge. The bridge master interface drives 0x2C, which is within the address range of the slave, and the transfer completes.

#### 12.5.4.4 Address Coherency

To simplify the system design, all masters should access slaves at the same location. In many systems, a processor passes buffer locations to other mastering components, such as a DMA controller. If the processor and DMA controller do not access the slave at the same location, Platform Designer must compensate for the differences.

**Figure 261. Slaves at Different Addresses and Complicating the System**

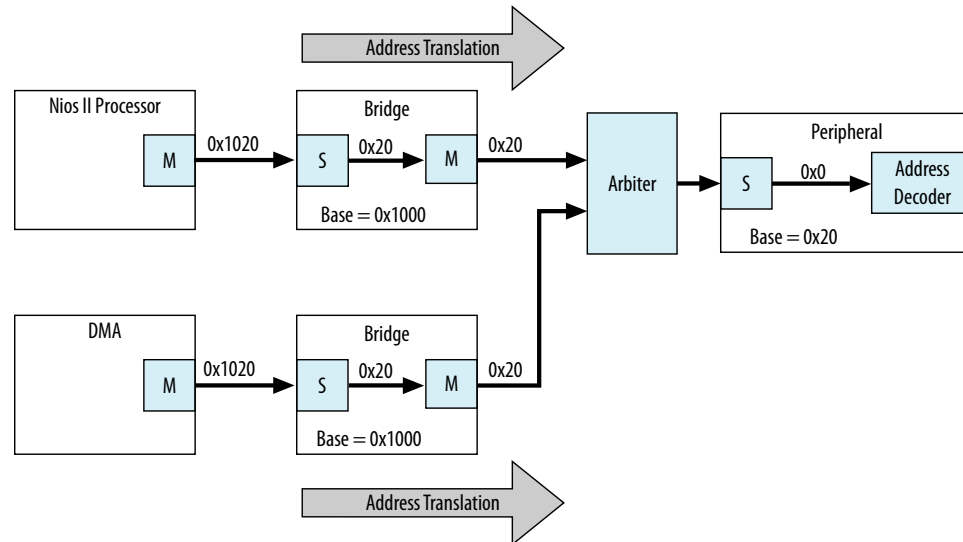


A Nios II processor and DMA controller access a slave interface located at address 0x20. The processor connects directly to the slave interface. The DMA controller connects to a pipeline bridge located at address 0x1000, which then connects to the slave interface. Because the DMA controller accesses the pipeline bridge first, it must drive 0x1020 to access the first location of the slave interface. Because the processor accesses the slave from a different location, you must maintain two base addresses for the slave device.

To avoid the requirement for two addresses, you can add an additional bridge to the system, set its base address to 0x1000, and then disable all the pipelining options in the second bridge so that the bridge has minimal impact on system timing and

resource utilization. Because this second bridge has the same base address as the original bridge, the processor and DMA controller access the slave interface with the same address range.

**Figure 262. Address Translation Corrected With Bridge**



## 12.6 Increasing Transfer Throughput

Increasing the transfer efficiency of the master and slave interfaces in your system increases the throughput of your design. Designs with strict cost or power requirements benefit from increasing the transfer efficiency because you can then use less expensive, lower frequency devices. Designs requiring high performance also benefit from increased transfer efficiency because increased efficiency improves the performance of frequency-limited hardware.

Throughput is the number of symbols (such as bytes) of data that Platform Designer can transfer in a given clock cycle. Read latency is the number of clock cycles between the address and data phase of a transaction. For example, a read latency of two means that the data is valid two cycles after the address is posted. If the master must wait for one request to finish before the next begins, such as with a processor, then the read latency is very important to the overall throughput.

You can measure throughput and latency in simulation by observing the waveforms, or using the verification IP monitors.

### Related Links

- [Avalon Verification IP Suite User Guide](#)
- [Mentor Graphics Verification IP Altera Edition AMBA 3 AXI and AMBA 4 AXI User Guide](#)



## 12.6.1 Using Pipelined Transfers

Pipelined transfers increase the read efficiency by allowing a master to post multiple reads before data from an earlier read returns. Masters that support pipelined transfers post transfers continuously, relying on the `readdatavalid` signal to indicate valid data. Slaves support pipelined transfers by including the `readdatavalid` signal or operating with a fixed read latency.

AXI masters declare how many outstanding writes and reads it can issue with the `writeIssuingCapability` and `readIssuingCapability` parameters. In the same way, a slave can declare how many reads it can accept with the `readAcceptanceCapability` parameter. AXI masters with a read issuing capability greater than one are pipelined in the same way as Avalon masters and the `readdatavalid` signal.

### 12.6.1.1 Using the Maximum Pending Reads Parameter

If you create a custom component with a slave interface supporting variable-latency reads, you must specify the **Maximum Pending Reads** parameter in the Component Editor. Platform Designer uses this parameter to generate the appropriate interconnect and represent the maximum number of read transfers that your pipelined slave component can process. If the number of reads presented to the slave interface exceeds the **Maximum Pending Reads** parameter, then the slave interface must assert `waitrequest`.

Optimizing the value of the **Maximum Pending Reads** parameter requires an understanding of the latencies of your custom components. This parameter should be based on the component's highest read latency for the various logic paths inside the component. For example, if your pipelined component has two modes, one requiring two clock cycles and the other five, set the **Maximum Pending Reads** parameter to 5 to allow your component to pipeline five transfers, and eliminating dead cycles after the initial five-cycle latency.

You can also determine the correct value for the **Maximum Pending Reads** parameter by monitoring the number of reads that are pending during system simulation or while running the hardware. To use this method, set the parameter to a high value and use a master that issues read requests on every clock. You can use a DMA for this task if the data is written to a location that does not frequently assert `waitrequest`. If you implement this method, you can observe your component with a logic analyzer or built-in monitoring hardware.

Choosing the correct value for the **Maximum Pending Reads** parameter of your custom pipelined read component is important. If you underestimate the parameter value, you may cause a master interface to stall with a `waitrequest` until the slave responds to an earlier read request and frees a FIFO position.

The **Maximum Pending Reads** parameter controls the depth of the response FIFO inserted into the interconnect for each master connected to the slave. This FIFO does not use significant hardware resources. Overestimating the **Maximum Pending Reads** parameter results in a slight increase in hardware utilization. For these reasons, if you are not sure of the optimal value, you should overestimate this value.

If your system includes a bridge, you must set the **Maximum Pending Reads** parameter on the bridge as well. To allow maximum throughput, this value should be equal to or greater than the **Maximum Pending Reads** value for the connected slave that has the highest value. You can limit the maximum pending reads of a slave and

reduce the buffer depth by reducing the parameter value on the bridge if the high throughput is not required. If you do not know the **Maximum Pending Reads** value for all the slave components, you can monitor the number of reads that are pending during system simulation while running the hardware. To use this method, set the **Maximum Pending Reads** parameter to a high value and use a master that issues read requests on every clock, such as a DMA. Then, reduce the number of maximum pending reads of the bridge until the bridge reduces the performance of any masters accessing the bridge.

## 12.6.2 Arbitration Shares and Bursts

Arbitration shares provide control over the arbitration process. By default, the arbitration algorithm allocates evenly, with all masters receiving one share.

You can adjust the arbitration process by assigning a larger number of shares to the masters that need greater throughput. The larger the arbitration share, the more transfers are allocated to the master to access a slave. The master gets uninterrupted access to the slave for its number of shares, as long as the master is reading or writing.

If a master cannot post a transfer and other masters are waiting to gain access to a particular slave, the arbiter grants another master access. This mechanism prevents a master from wasting arbitration cycles if it cannot post back-to-back transfers. A bursting transaction contains multiple beats (or words) of data, starting from a single address. Bursts allow a master to maintain access to a slave for more than a single word transfer. If a bursting master posts a write transfer with a burst length of eight, it is guaranteed arbitration for eight write cycles.

You can assign arbitration shares to an Avalon-MM bursting master and AXI masters (which are always considered a bursting master). Each share consists of one burst transaction (such as multi-cycle write), and allows a master to complete a number of bursts before arbitration switches to the next master.

### Related Links

[Arbitration](#) on page 671

### 12.6.2.1 Differences Between Arbitration Shares and Bursts

The following three key characteristics distinguish arbitration shares and bursts:

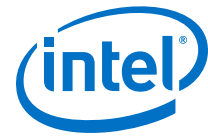
- Arbitration Lock
- Sequential Addressing
- Burst Adapters

#### Arbitration Lock

When a master posts a burst transfer, the arbitration is locked for that master; consequently, the bursting master should be capable of sustaining transfers for the duration of the locked period. If, after the fourth write, the master deasserts the write signal (Avalon-MM write or AXI `wvalid`) for fifty cycles, all other masters continue to wait for access during this stalled period.

To avoid wasted bandwidth, your master designs should wait until a full burst transfer is ready before requesting access to a slave device. Alternatively, you can avoid wasted bandwidth by posting `burstcounts` equal to the amount of data that is ready.





For example, if you create a custom bursting write master with a maximum `burstcount` of eight, but only three words of data are ready, you can present a `burstcount` of three. This strategy does not result in optimal use of the system band width if the slave is capable of handling a larger burst; however, this strategy prevents stalling and allows access for other masters in the system.

### Sequential Addressing

An Avalon-MM burst transfer includes a base address and a `burstcount`, which represents the number of words of data that are transferred, starting from the base address and incrementing sequentially. Burst transfers are common for processors, DMAs, and buffer processing accelerators; however, sometimes a master must access non-sequential addresses. Consequently, a bursting master must set the `burstcount` to the number of sequential addresses, and then reset the `burstcount` for the next location.

The arbitration share algorithm has no restrictions on addresses; therefore, your custom master can update the address it presents to the interconnect for every read or write transaction.

### Burst Adapters

Platform Designer allows you to create systems that mix bursting and non-bursting master and slave interfaces. This design strategy allows you to connect bursting master and slave interfaces that support different maximum burst lengths, with Platform Designer generating burst adapters when appropriate.

Platform Designer inserts a burst adapter whenever a master interface burst length exceeds the burst length of the slave interface, or if the master issues a burst type that the slave cannot support. For example, if you connect an AXI master to an Avalon slave, a burst adapter is inserted. Platform Designer assigns non-bursting masters and slave interfaces a burst length of one. The burst adapter divides long bursts into shorter bursts. As a result, the burst adapter adds logic to the address and `burstcount` paths between the master and slave interfaces.

## 12.6.2.2 Choosing Avalon-MM Interface Types

To avoid inefficient Avalon-MM transfers, custom master or slave interfaces must use the appropriate simple, pipelined, or burst interfaces.

### 12.6.2.2.1 Simple Avalon-MM Interfaces

Simple interface transfers do not support pipelining or bursting for reads or writes; consequently, their performance is limited. Simple interfaces are appropriate for transfers between masters and infrequently used slave interfaces. In Platform Designer, the PIO, UART, and Timer include slave interfaces that use simple transfers.

### 12.6.2.2.2 Pipelined Avalon-MM Interfaces

Pipelined read transfers allow a pipelined master interface to start multiple read transfers in succession without waiting for prior transfers to complete. Pipelined transfers allow master-slave pairs to achieve higher throughput, even though the slave port may require one or more cycles of latency to return data for each transfer.

In many systems, read throughput becomes inadequate if simple reads are used and pipelined transfers can increase throughput. If you define a component with a fixed read latency, Platform Designer automatically provides the pipelining logic necessary to support pipelined reads. You can use fixed latency pipelining as the default design starting point for slave interfaces. If your slave interface has a variable latency response time, use the `readdatavalid` signal to indicate when valid data is available. The interconnect implements read response FIFO buffering to handle the maximum number of pending read requests.

To use components that support pipelined read transfers, and to use a pipelined system interconnect efficiently, your system must contain pipelined masters. You can use pipelined masters as the default starting point for new master components. Use the `readdatavalid` signal for these master interfaces.

Because master and slaves sometimes have mismatched pipeline latency, the interconnect contains logic to reconcile the differences.

**Table 188. Pipeline Latency in a Master-Slave Pair**

Master	Slave	Pipeline Management Logic Structure
No pipeline	No pipeline	Platform Designer interconnect does not instantiate logic to handle pipeline latency.
No pipeline	Pipelined with fixed or variable latency	Platform Designer interconnect forces the master to wait through any slave-side latency cycles. This master-slave pair gains no benefits from pipelining, because the master waits for each transfer to complete before beginning a new transfer. However, while the master is waiting, the slave can accept transfers from a different master.
Pipelined	No pipeline	Platform Designer interconnect carries out the transfer as if neither master nor slave were pipelined, causing the master to wait until the slave returns data. An example of a non-pipeline slave is an asynchronous off-chip interface.
Pipelined	Pipelined with fixed latency	Platform Designer interconnect allows the master to capture data at the exact clock cycle when data from the slave is valid, to enable maximum throughput. An example of a fixed latency slave is an on-chip memory.
Pipelined	Pipelined with variable latency	The slave asserts a signal when its <code>readdata</code> is valid, and the master captures the data. The master-slave pair can achieve maximum throughput if the slave has variable latency. Examples of variable latency slaves include SDRAM and FIFO memories.

### 12.6.2.2.3 Burst Avalon-MM Interfaces

Burst transfers are commonly used for latent memories such as SDRAM and off-chip communication interfaces, such as PCI Express. To use a burst-capable slave interface efficiently, you must connect to a bursting master. Components that require bursting to operate efficiently typically have an overhead penalty associated with short bursts or non-bursting transfers.

You can use a burst-capable slave interface if you know that your component requires sequential transfers to operate efficiently. Because SDRAM memories incur a penalty when switching banks or rows, performance improves when SDRAM memories are accessed sequentially with bursts.

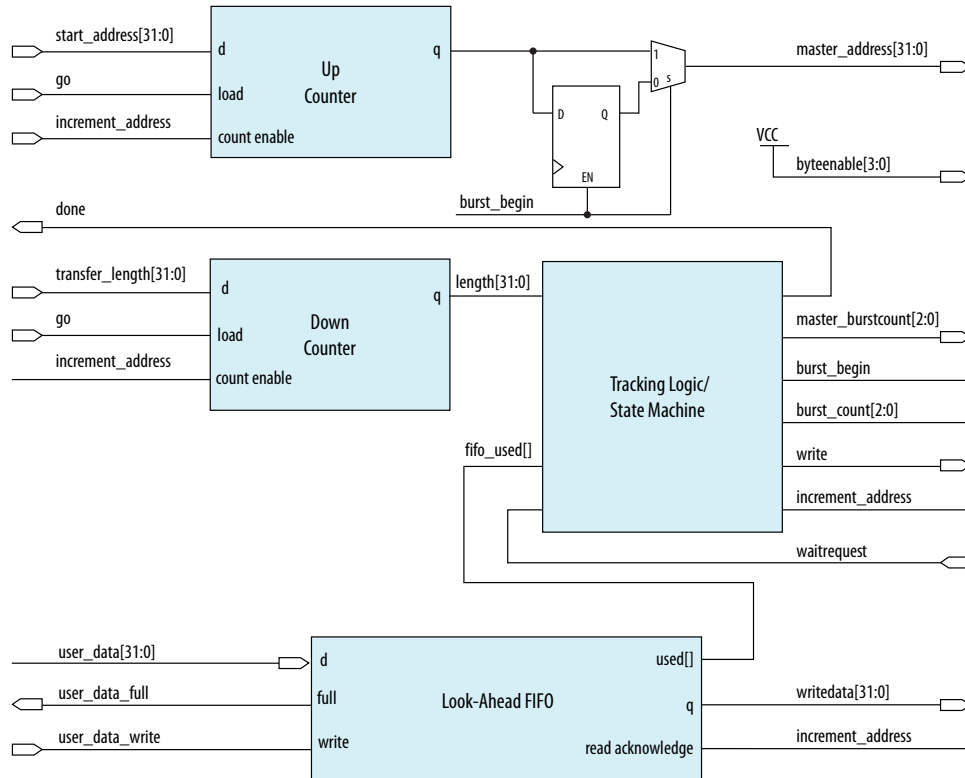
Architectures that use the same signals to transfer address and data also benefit from bursting. Whenever an address is transferred over shared address and data signals, the throughput of the data transfer is reduced. Because the address phase adds overhead, using large bursts increases the throughput of the connection.



### 12.6.2.3 Avalon-MM Burst Master Example

**Figure 263. Avalon Bursting Write Master**

This example shows the architecture of a bursting write master that receives data from a FIFO and writes the contents to memory. You can use a bursting master as a starting point for your own bursting components, such as custom DMAs, hardware accelerators, or off-chip communication interfaces.



The master performs word accesses and writes to sequential memory locations. When `go` is asserted, the `start_address` and `transfer_length` are registered. On the next clock cycle, the control logic asserts `burst_begin`, which synchronizes the internal control signals in addition to the `master_address` and `master_burstcount` presented to the interconnect. The timing of these two signals is important because during bursting write transfers `byteenable` and `burstcount` must be held constant for the entire burst.

To avoid inefficient writes, the master posts a burst when enough data is buffered in the FIFO. To maximize the burst efficiency, the master should stall only when a slave asserts `waitrequest`. In this example, the FIFO's `used` signal tracks the number of words of data that are stored in the FIFO and determines when enough data has been buffered.

The `address` register increments after every word transfer, and the `length` register decrements after every word transfer. The address remains constant throughout the burst. Because a transfer is not guaranteed to complete on burst boundaries, additional logic is necessary to recognize the completion of short bursts and complete the transfer.

### Related Links

[Avalon Memory-Mapped Master Templates](#)

## 12.7 Reducing Logic Utilization

You can minimize logic size of Platform Designer systems. Typically, there is a trade-off between logic utilization and performance. Reducing logic utilization applies to both Avalon and AXI interfaces.

### 12.7.1 Minimizing Interconnect Logic to Reduce Logic Utilization

In Platform Designer, changes to the connections between master and slave reduce the amount of interconnect logic required in the system.

#### Related Links

[Limited Concurrency](#) on page 762

#### 12.7.1.1 Creating Dedicated Master and Slave Connections to Minimize Interconnect Logic

You can create a system where a master interface connects to a single slave interface. This configuration eliminates address decoding, arbitration, and return data multiplexing, which simplifies the interconnect. Dedicated master-to-slave connections attain the same clock frequencies as Avalon-ST connections.

Typically, these one-to-one connections include an Avalon memory-mapped bridge or hardware accelerator. For example, if you insert a pipeline bridge between a slave and all other master interfaces, the logic between the bridge master and slave interface is reduced to wires. If a hardware accelerator connects only to a dedicated memory, no system interconnect logic is generated between the master and slave pair.

#### 12.7.1.2 Removing Unnecessary Connections to Minimize Interconnect Logic

The number of connections between master and slave interfaces affects the  $f_{MAX}$  of your system. Every master interface that you connect to a slave interface increases the width of the multiplexer width. As a multiplexer width increases, so does the logic depth and width that implements the multiplexer in the FPGA. To improve system performance, connect masters and slaves only when necessary.

When you connect a master interface to many slave interfaces, the multiplexer for the read data signal grows. Avalon typically uses a `readdata` signal. AXI read data signals add a response status and last indicator to the read response channel using `rdata`, `rresp`, and `rlast`. Additionally, bridges help control the depth of multiplexers.

#### Related Links

[Implementing Command Pipelining \(Master-to-Slave\)](#) on page 755

#### 12.7.1.3 Simplifying Address Decode Logic

If address code logic is in the critical path, you may be able to change the address map to simplify the decode logic. Experiment with different address maps, including a one-hot encoding, to see if results improve.



## 12.7.2 Minimizing Arbitration Logic by Consolidating Multiple Interfaces

As the number of components in a design increases, the amount of logic required to implement the interconnect also increases. The number of arbitration blocks increases for every slave interface that is shared by multiple master interfaces. The width of the read data multiplexer increases as the number of slave interfaces supporting read transfers increases on a per master interface basis. For these reasons, consider implementing multiple blocks of logic as a single interface to reduce interconnect logic utilization.

### 12.7.2.1 Logic Consolidation Trade-Offs

You should consider the following trade-offs before making modifications to your system or interfaces:

- Consider the impact on concurrency that results when you consolidate components. When a system has four master components and four slave interfaces, it can initiate four concurrent accesses. If you consolidate the four slave interfaces into a single interface, then the four masters must compete for access. Consequently, you should only combine low priority interfaces such as low speed parallel I/O devices if the combination does not impact the performance.
- Determine whether consolidation introduces new decode and multiplexing logic for the slave interface that the interconnect previously included. If an interface contains multiple read and write address locations, the interface already contains the necessary decode and multiplexing logic. When you consolidate interfaces, you typically reuse the decoder and multiplexer blocks already present in one of the original interfaces; however, combining interfaces may simply move the decode and multiplexer logic, rather than eliminate duplication.
- Consider whether consolidating interfaces makes the design complicated. If so, you should not consolidate interfaces.

#### Related Links

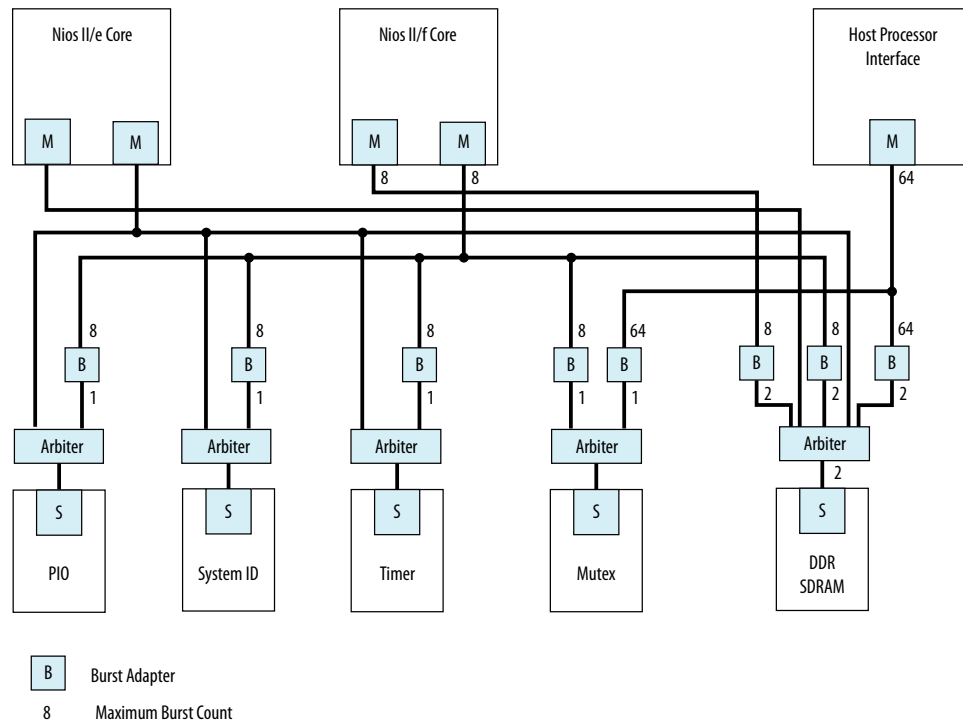
[Using Concurrency in Memory-Mapped Systems](#) on page 748

### 12.7.2.2 Consolidating Interfaces

In this example, we have a system with a mix of components, each having different burst capabilities: a Nios II/e core, a Nios II/f core, and an external processor, which off-loads some processing tasks to the Nios II/f core.

The Nios II/f core supports a maximum burst size of eight. The external processor interface supports a maximum burst length of 64. The Nios II/e core does not support bursting. The memory in the system is SDRAM with an Avalon maximum burst length of two.

Figure 264. Mixed Bursting System



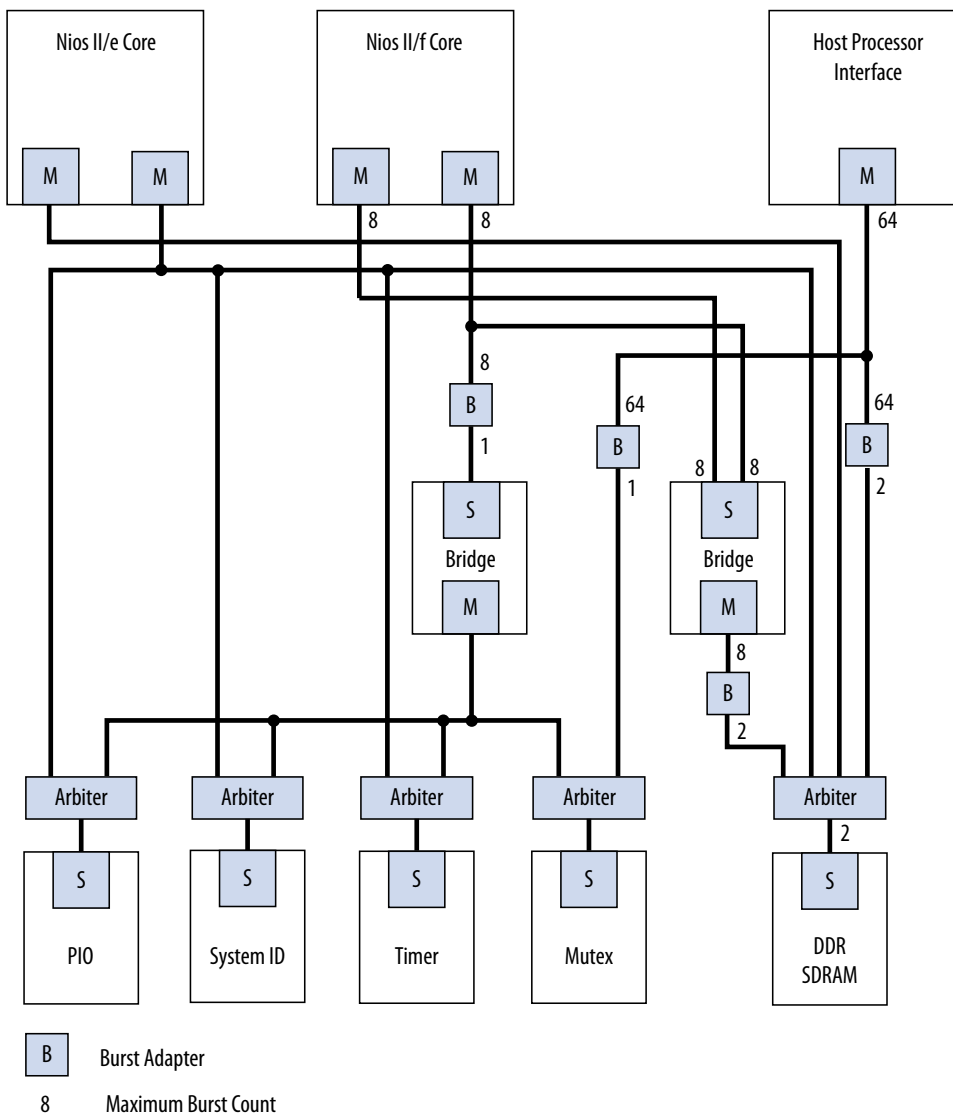
Platform Designer automatically inserts burst adapters to compensate for burst length mismatches. The adapters reduce bursts to a single transfer, or the length of two transfers. For the external processor interface connecting to DDR SDRAM, a burst of 64 words is divided into 32 burst transfers, each with a burst length of two. When you generate a system, Platform Designer inserts burst adapters based on maximum `burstcount` values; consequently, the interconnect logic includes burst adapters between masters and slave pairs that do not require bursting, if the master is capable of bursts.

In this example, Platform Designer inserts a burst adapter between the Nios II processors and the timer, system ID, and PIO peripherals. These components do not support bursting and the Nios II processor performs a single word read and write accesses to these components.



**Figure 265. Mixed Bursting System with Bridges**

To reduce the number of adapters, you can add pipeline bridges. The pipeline bridge, between the Nios II/f core and the peripherals that do not support bursts, eliminates three burst adapters from the previous example. A second pipeline bridge between the Nios II/f core and the DDR SDRAM, with its maximum burst size set to eight, eliminates another burst adapter, as shown below.



### 12.7.3 Reducing Logic Utilization With Multiple Clock Domains

You specify clock domains in Platform Designer on the **System Contents** tab. Clock sources can be driven by external input signals to Platform Designer, or by PLLs inside Platform Designer. Clock domains are differentiated based on the name of the clock. You can create multiple asynchronous clocks with the same frequency.

Platform Designer generates Clock Domain Crossing (CDC) logic that hides the details of interfacing components operating in different clock domains. The interconnect supports the memory-mapped protocol with each port independently, and therefore

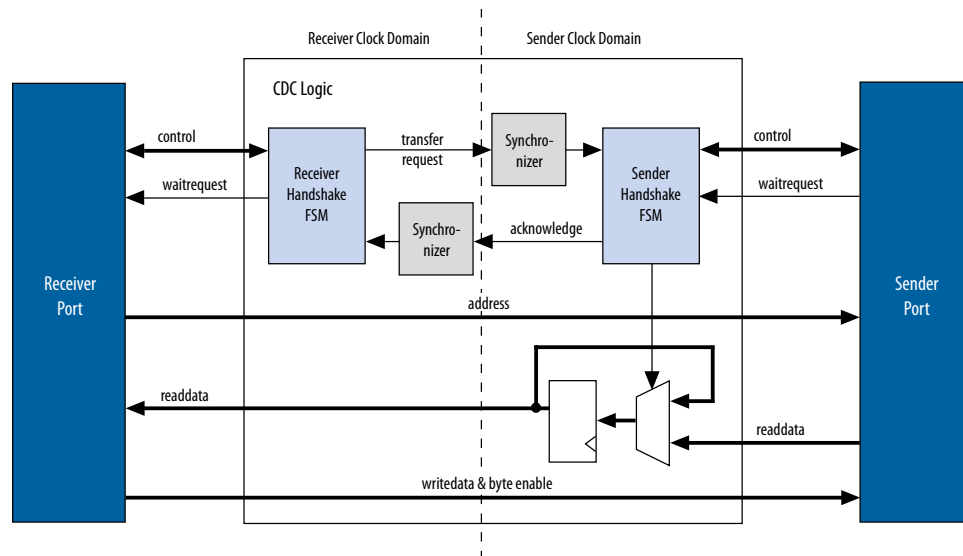
masters do not need to incorporate clock adapters in order to interface to slaves on a different domain. Platform Designer interconnect logic propagates transfers across clock domain boundaries automatically.

Clock-domain adapters provide the following benefits:

- Allows component interfaces to operate at different clock frequencies.
- Eliminates the need to design CDC hardware.
- Allows each memory-mapped port to operate in only one clock domain, which reduces design complexity of components.
- Enables masters to access any slave without communication with the slave clock domain.
- Allows you to focus performance optimization efforts on components that require fast clock speed.

A clock domain adapter consists of two finite state machines (FSM), one in each clock domain, that use a hand-shaking protocol to propagate transfer control signals (`read_request`, `write_request`, and the master `waitrequest` signals) across the clock boundary.

**Figure 266. Clock Crossing Adapter**



This example illustrates a clock domain adapter between one master and one slave. The synchronizer blocks use multiple stages of flipflops to eliminate the propagation of meta-stable events on the control signals that enter the handshake FSMs. The CDC logic works with any clock ratio.





The typical sequence of events for a transfer across the CDC logic is as follows:

- The master asserts address, data, and control signals.
- The master handshake FSM captures the control signals and immediately forces the master to wait. The FSM uses only the control signals, not address and data. For example, the master simply holds the address signal constant until the slave side has safely captured it.
- The master handshake FSM initiates a transfer request to the slave handshake FSM.
- The transfer request is synchronized to the slave clock domain.
- The slave handshake FSM processes the request, performing the requested transfer with the slave.
- When the slave transfer completes, the slave handshake FSM sends an acknowledge back to the master handshake FSM. The acknowledge is synchronized back to the master clock domain.
- The master handshake FSM completes the transaction by releasing the master from the wait condition.

Transfers proceed as normal on the slave and the master side, without a special protocol to handle crossing clock domains. From the perspective of a slave, there is nothing different about a transfer initiated by a master in a different clock domain. From the perspective of a master, a transfer across clock domains simply requires extra clock cycles. Similar to other transfer delay cases (for example, arbitration delay or wait states on the slave side), the Platform Designer forces the master to wait until the transfer terminates. As a result, pipeline master ports do not benefit from pipelining when performing transfers to a different clock domain.

Platform Designer automatically determines where to insert CDC logic based on the system and the connections between components, and places CDC logic to maintain the highest transfer rate for all components. Platform Designer evaluates the need for CDC logic for each master and slave pair independently, and generates CDC logic wherever necessary.

#### **Related Links**

[Avalon Memory-Mapped Design Optimizations](#)

### **12.7.4 Duration of Transfers Crossing Clock Domains**

CDC logic extends the duration of master transfers across clock domain boundaries. In the worst case, which is for reads, each transfer is extended by five master clock cycles and five slave clock cycles. Assuming the default value of 2 for the master domain synchronizer length and the slave domain synchronizer length, the components of this delay are the following:

- Four additional master clock cycles, due to the master-side clock synchronizer.
- Four additional slave clock cycles, due to the slave-side clock synchronizer.
- One additional clock in each direction, due to potential metastable events as the control signals cross clock domains.



**Note:** Systems that require a higher performance clock should use the Avalon-MM clock crossing bridge instead of the automatically inserted CDC logic. The clock crossing bridge includes a buffering mechanism so that multiple reads and writes can be pipelined. After paying the initial penalty for the first read or write, there is no additional latency penalty for pending reads and writes, increasing throughput by up to four times, at the expense of added logic resources.

## 12.8 Reducing Power Consumption

Platform Designer provides various low power design changes that enable you to reduce the power consumption of the interconnect and custom components.

### 12.8.1 Reducing Power Consumption With Multiple Clock Domains

When you use multiple clock domains, you should put non-critical logic in the slower clock domain. Platform Designer automatically reconciles data crossing over asynchronous clock domains by inserting clock crossing logic (handshake or FIFO).

You can use clock crossing in Platform Designer to reduce the clock frequency of the logic that does not require a high frequency clock, which allows you to reduce power consumption. You can use either handshaking clock crossing bridges or handshaking clock crossing adapters to separate clock domains.

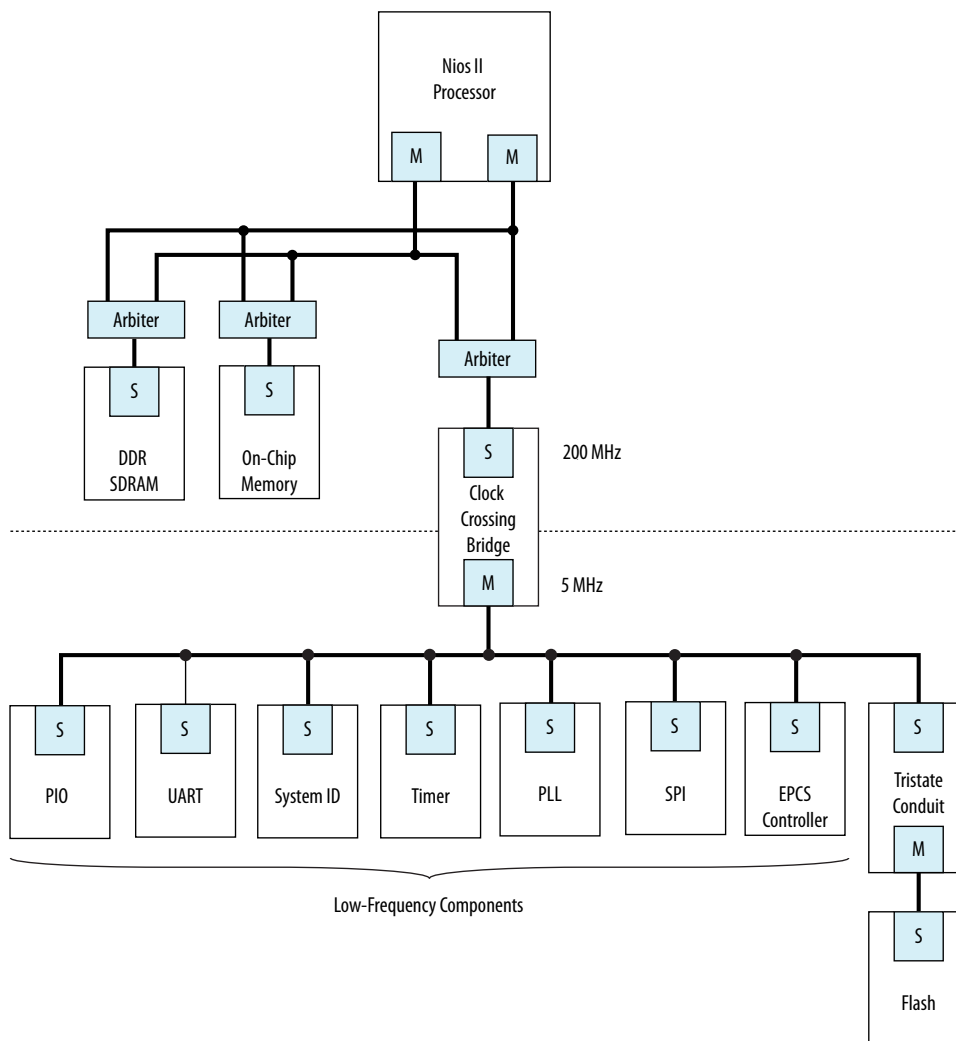
You can use the clock crossing bridge to connect master interfaces operating at a higher frequency to slave interfaces running at a lower frequency. Only connect low throughput or low priority components to a clock crossing bridge that operates at a reduced clock frequency. The following are examples of low throughput or low priority components:

- PIOs
- UARTs (JTAG or RS-232)
- System identification (SysID)
- Timers
- PLL (instantiated within Platform Designer)
- Serial peripheral interface (SPI)
- EPCS controller
- Tristate bridge and the components connected to the bridge

By reducing the clock frequency of the components connected to the bridge, you reduce the dynamic power consumption of the design. Dynamic power is a function of toggle rates and decreasing the clock frequency decreases the toggle rate.



Figure 267. Reducing Power Utilization Using a Bridge to Separate Clock Domains



Platform Designer automatically inserts clock crossing adapters between master and slave interfaces that operate at different clock frequencies. You can choose the type of clock crossing adapter in the Platform Designer **Project Settings** tab. Adapters do not appear in the **Connections** column because you do not insert them. The following clock crossing adapter types are available in Platform Designer:

- **Handshake**—Uses a simple handshaking protocol to propagate transfer control signals and responses across the clock boundary. This adapter uses fewer hardware resources because each transfer is safely propagated to the target domain before the next transfer begins. The Handshake adapter is appropriate for systems with low throughput requirements.
- **FIFO**—Uses dual-clock FIFOs for synchronization. The latency of the FIFO adapter is approximately two clock cycles more than the handshake clock crossing component, but the FIFO-based adapter can sustain higher throughput because it supports multiple transactions simultaneously. The FIFO adapter requires more resources, and is appropriate for memory-mapped transfers requiring high throughput across clock domains.
- **Auto**—Platform Designer specifies the appropriate FIFO adapter for bursting links and the Handshake adapter for all other links.

Because the clock crossing bridge uses FIFOs to implement the clock crossing logic, it buffers transfers and data. Clock crossing adapters are not pipelined, so that each transaction is blocking until the transaction completes. Blocking transactions may lower the throughput substantially; consequently, if you want to reduce power consumption without limiting the throughput significantly, you should use the clock crossing bridge or the FIFO clock crossing adapter. However, if the design requires single read transfers, a clock crossing adapter is preferable because the latency is lower.

The clock crossing bridge requires few logic resources other than on-chip memory. The number of on-chip memory blocks used is proportional to the address span, data width, buffering depth, and bursting capabilities of the bridge. The clock crossing adapter does not use on-chip memory and requires a moderate number of logic resources. The address span, data width, and the bursting capabilities of the clock crossing adapter determine the resource utilization of the device.

When you decide to use a clock crossing bridge or clock crossing adapter, you must consider the effects of throughput and memory utilization in the design. If on-chip memory resources are limited, you may be forced to choose the clock crossing adapter. Using the clock crossing bridge to reduce the power of a single component may not justify using more resources. However, if you can place all of the low priority components behind a single clock crossing bridge, you may reduce power consumption in the design.

### Related Links

[Power Optimization](#)



## 12.8.2 Reducing Power Consumption by Minimizing Toggle Rates

A Platform Designer system consumes power whenever logic transitions between on and off states. When the state is held constant between clock edges, no charging or discharging occurs. You can use the following design methodologies to reduce the toggle rates of your design:

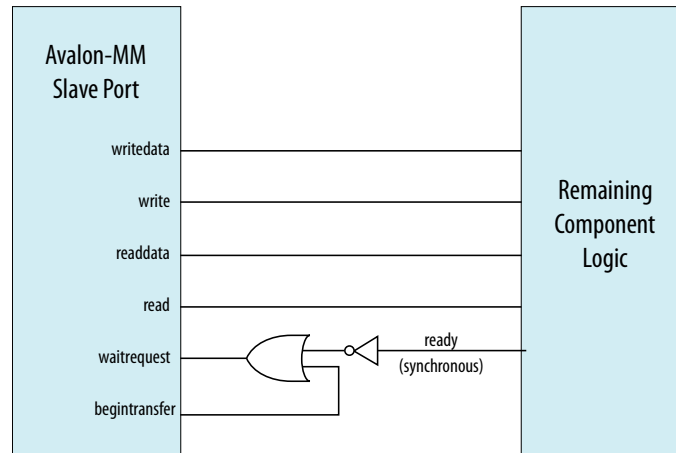
- Registering component boundaries
- Using clock enable signals
- Inserting bridges

Platform Designer interconnect is uniquely combinational when no adapters or bridges are present and there is no interconnect pipelining. When a slave interface is not selected by a master, various signals may toggle and propagate into the component. By registering the boundary of your component at the master or slave interface, you can minimize the toggling of the interconnect and your component. In addition, registering boundaries can improve operating frequency. When you register the signals at the interface level, you must ensure that the component continues to operate within the interface standard specification.

Avalon-MM `waitrequest` is a difficult signal to synchronize when you add registers to your component. The `waitrequest` signal must be asserted during the same clock cycle that a master asserts read or write to in order to prolong the transfer. A master interface can read the `waitrequest` signal too early and post more reads and writes prematurely.

*Note:* There is no direct AXI equivalent for `waitrequest` and `burstcount`, though the *AMBA Protocol Specification* implies that the AXI `ready` signal cannot depend combinatorially on the AXI `valid` signal. Therefore, Platform Designer typically buffers AXI component boundaries for the `ready` signal.

For slave interfaces, the interconnect manages the `begintransfer` signal, which is asserted during the first clock cycle of any read or write transfer. If the `waitrequest` is one clock cycle late, you can logically OR the `waitrequest` and the `begintransfer` signals to form a new `waitrequest` signal that is properly synchronized. Alternatively, the component can assert `waitrequest` before it is selected, guaranteeing that the `waitrequest` is already asserted during the first clock cycle of a transfer.

**Figure 268. Variable Latency**


### Using Clock Enables

You can use clock enables to hold the logic in a steady state, and the `write` and `read` signals as clock enables for slave components. Even if you add registers to your component boundaries, the interface can potentially toggle without the use of clock enables. You can also use the clock enable to disable combinational portions of the component.

For example, you can use an active high clock enable to mask the inputs into the combinational logic to prevent it from toggling when the component is inactive. Before preventing inactive logic from toggling, you must determine if the masking causes the circuit to function differently. If masking causes a functional failure, it may be possible to use a register stage to hold the combinational logic constant between clock cycles.

### Inserting Bridges

You can use bridges to reduce toggle rates, if you do not want to modify the component by using boundary registers or clock enables. A bridge acts as a repeater where transfers to the slave interface are repeated on the master interface. If the bridge is not accessed, the components connected to its master interface are also not accessed. The master interface of the bridge remains idle until a master accesses the bridge slave interface.

Bridges can also reduce the toggle rates of signals that are inputs to other master interfaces. These signals are typically `readdata`, `readdatavalid`, and `waitrequest`. Slave interfaces that support read accesses drive the `readdata`, `readdatavalid`, and `waitrequest` signals. A bridge inserts either a register or clock crossing FIFO between the slave interface and the master to reduce the toggle rate of the master input signals.

## 12.8.3 Reducing Power Consumption by Disabling Logic

There are typically two types of low power modes: volatile and non-volatile. A volatile low power mode holds the component in a reset state. When the logic is reactivated, the previous operational state is lost. A non-volatile low power mode restores the previous operational state. You can use either software-controlled or hardware-controlled sleep modes to disable a component in order to reduce power consumption.



### Software-Controlled Sleep Mode

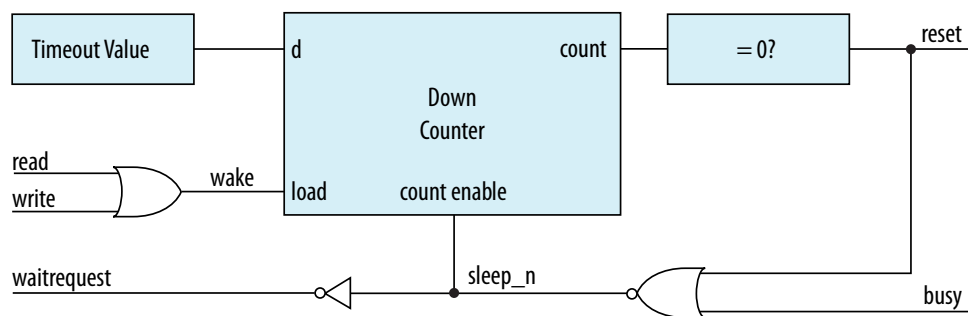
To design a component that supports software-controlled sleep mode, create a single memory-mapped location that enables and disables logic by writing a zero or one. You can use the register's output as a clock enable or reset, depending on whether the component has non-volatile requirements. The slave interface must remain active during sleep mode so that the enable bit is set when the component needs to be activated.

If multiple masters can access a component that supports sleep mode, you can use the mutex core to provide mutually exclusive accesses to your component. You can also build in the logic to re-enable the component on the very first access by any master in your system. If the component requires multiple clock cycles to re-activate, then it must assert a wait request to prolong the transfer as it exits sleep mode.

### Hardware-Controlled Sleep Mode

Alternatively, you can implement a timer in your component that automatically causes the component to enter a sleep mode based on a timeout value specified in clock cycles between read or write accesses. Each access resets the timer to the timeout value. Each cycle with no accesses decrements the timeout value by one. If the counter reaches zero, the hardware enters sleep mode until the next access.

Figure 269. Hardware-Controlled Sleep Components



This example provides a schematic for the hardware-controlled sleep mode. If restoring the component to an active state takes a long time, use a long timeout value so that the component is not continuously entering and exiting sleep mode. The slave interface must remain functional while the rest of the component is in sleep mode. When the component exits sleep mode, the component must assert the `waitrequest` signal until it is ready for read or write accesses.

### Related Links

[Mutex Core](#)

## 12.9 Reset Polarity and Synchronization in Platform Designer

When you add a component interface with a reset signal, Platform Designer defines its polarity as `reset(active-high)` or `reset_n(active-low)`.

You can view the polarity status of a reset signal by selecting the signal in the **Hierarchy** tab, and then view its expanded definition in the open **Parameters** and **Block Symbol** tabs. When you generate your component, Platform Designer interconnect automatically inverts polarities as needed.

Figure 270. Reset Signal (Active-High)

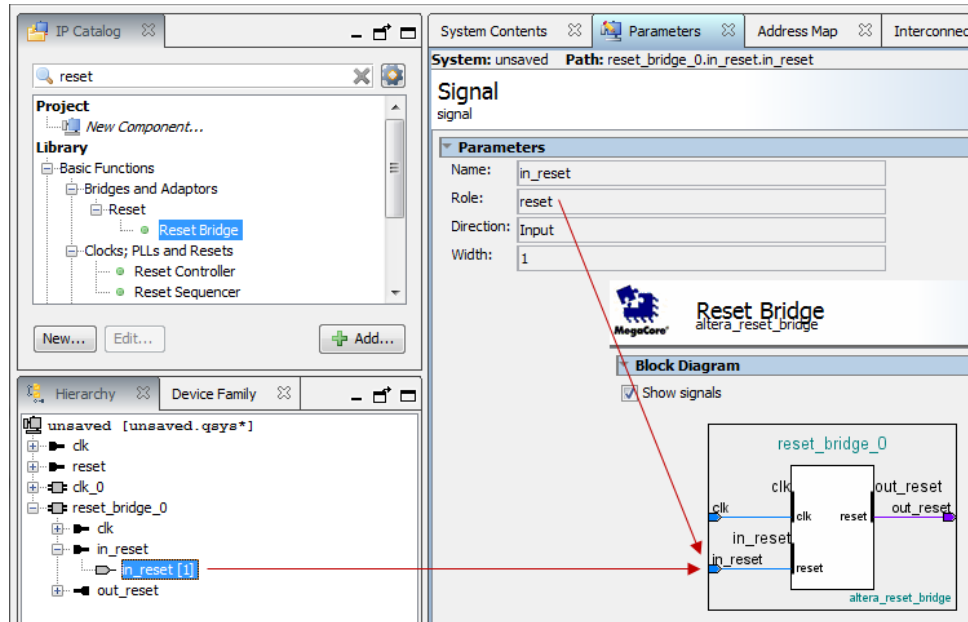
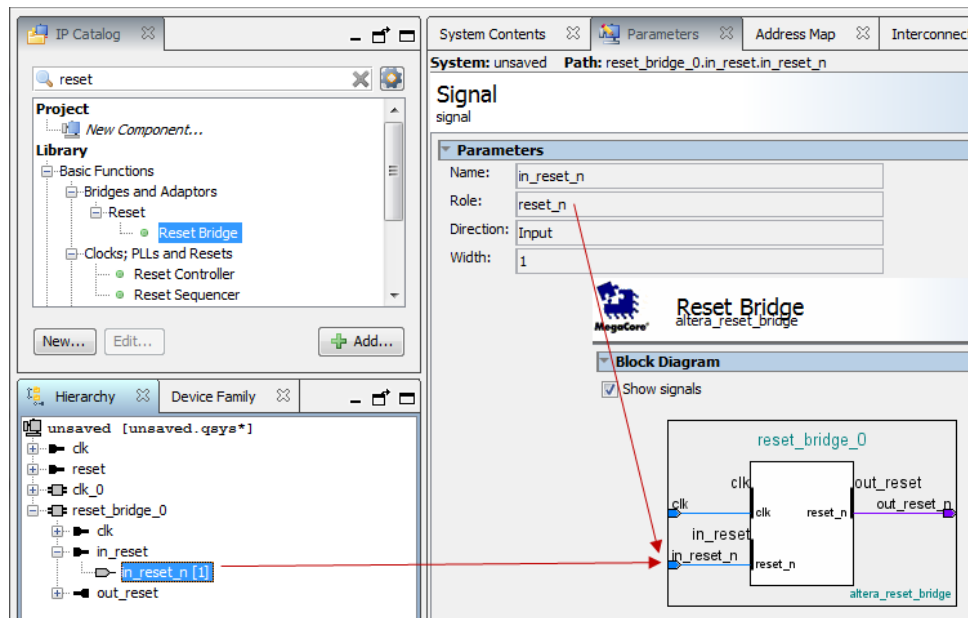


Figure 271. Reset Signal Active-Low



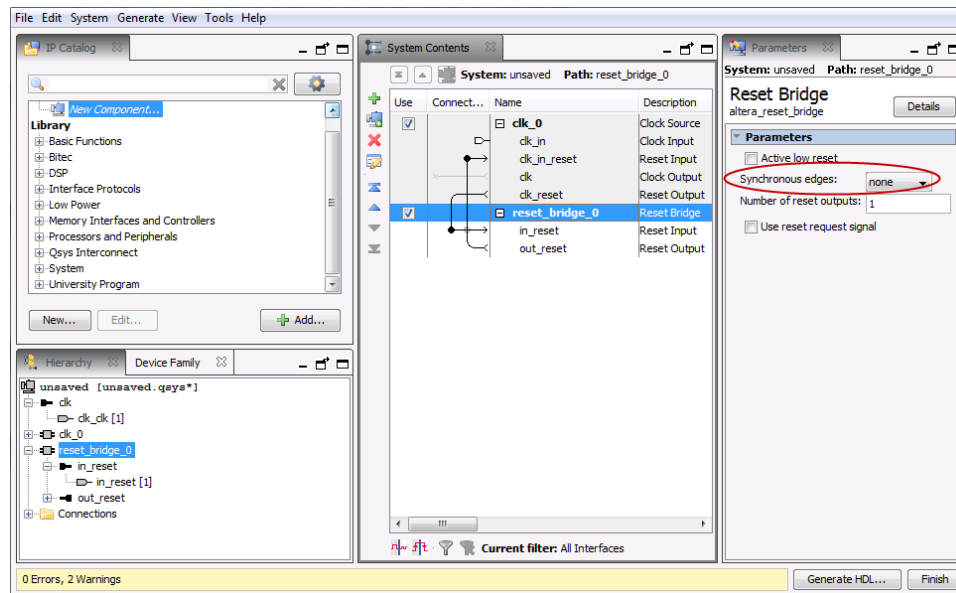
Each Platform Designer component has its own requirements for reset synchronization. Some blocks have internal synchronization and have no requirements, whereas other blocks require an externally synchronized reset. You can define how resets are synchronized in your Platform Designer system with the **Synchronous edges** parameter. In the clock source or reset bridge component, set the value of the **Synchronous edges** parameter to one of the following, depending on how the reset is externally synchronized:





- **None**—There is no synchronization on this reset.
- **Both**—The reset is synchronously asserted and deasserted with respect to the input clock.
- **Deassert**—The reset is synchronously asserted with respect to the input clock, and asynchronously deasserted.

Figure 272. Synchronous Edges Parameter



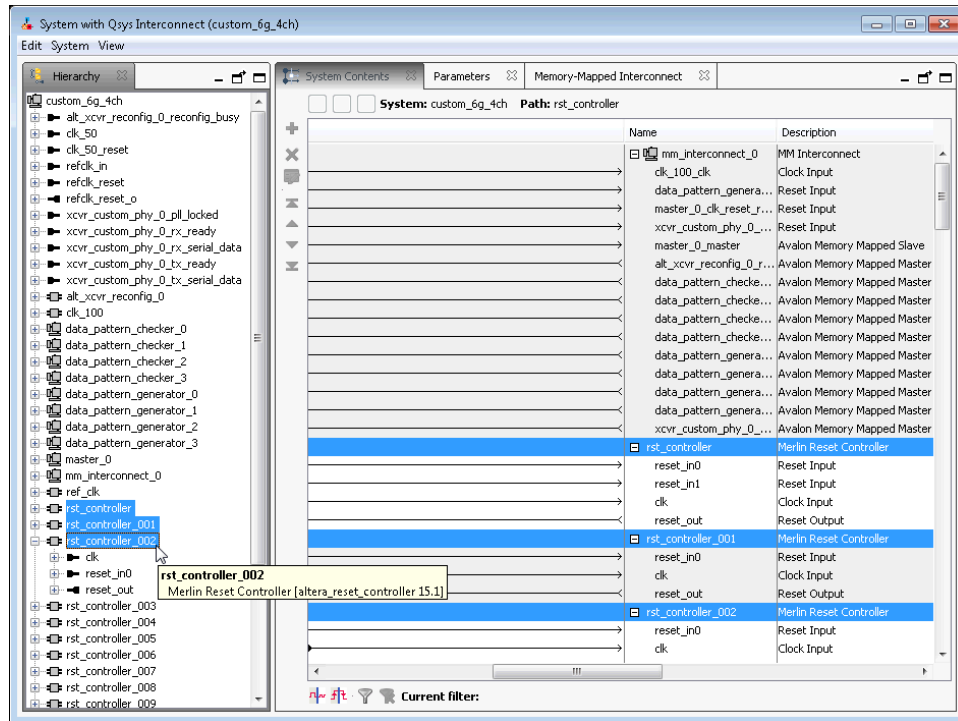
You can combine multiple reset sources to reset a particular component.

Figure 273. Combine Multiple Reset Sources

Use	Connections	Name	Description
<input checked="" type="checkbox"/>		<b>clk_0</b>	Clock Source
		clk_in	Clock Input
		clk_in_reset	Reset Input
		clk	Clock Output
		clk_reset	Reset Output
<input checked="" type="checkbox"/>		<b>reset_bridge_0</b>	Reset Bridge
		in_reset	Reset Input
		out_reset	Reset Output
<input checked="" type="checkbox"/>		<b>mm_bridge_0</b>	Avalon-MM Pipeline Bridge
		clk	Clock Input
	<b>reset</b>	Reset Input	
	s0	Avalon Memory Mapped Slave	
	m0	Avalon Memory Mapped Master	

When you generate your component, Platform Designer inserts adapters to synchronize or invert resets if there are mismatches in polarity or synchronization between the source and destination. You can view inserted adapters on the **Memory-Mapped Interconnect** tab with the **System > Show System with Platform Designer Interconnect** command.

Figure 274. Platform Designer Interconnect



## 12.10 Optimizing Platform Designer System Performance Design Examples

[Avalon Pipelined Read Master Example](#) on page 786

[Multiplexer Examples](#) on page 788

### 12.10.1 Avalon Pipelined Read Master Example

For a high throughput system using the Avalon-MM standard, you can design a pipelined read master that allows a system to issue multiple read requests before data returns. Pipelined read masters hide the latency of read operations by posting reads as frequently as every clock cycle. You can use this type of master when the address logic is not dependent on the data returning.

#### 12.10.1.1 Avalon Pipelined Read Master Example Design Requirements

You must carefully design the logic for the control and datapaths of pipelined read masters. The control logic must extend a read cycle whenever the `waitrequest` signal is asserted. This logic must also control the master address, `byteenable`,



and `read` signals. To achieve maximum throughput, pipelined read masters should post reads continuously while `waitrequest` is deasserted. While `read` is asserted, the address presented to the interconnect is stored.

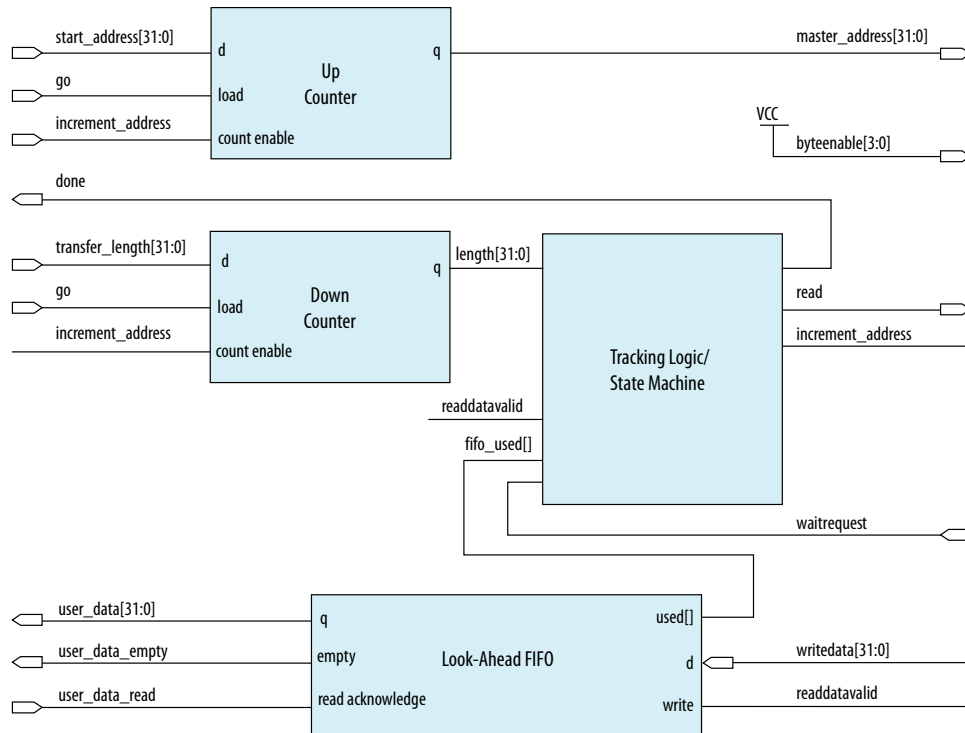
The datapath logic includes the `readdata` and `readdatavalid` signals. If your master can accept data on every clock cycle, you can register the data with the `readdatavalid` as an enable bit. If your master cannot process a continuous stream of read data, it must buffer the data in a FIFO. The control logic must stop issuing reads when the FIFO reaches a predetermined fill level to prevent FIFO overflow.

### 12.10.1.2 Expected Throughput Improvement

The throughput improvement that you can achieve with a pipelined read master is typically directly proportional to the pipeline depth of the interconnect and the slave interface. For example, if the total latency is two cycles, you can double the throughput by inserting a pipelined read master, assuming the slave interface also supports pipeline transfers. If either the master or slave does not support pipelined read transfers, then the interconnect asserts `waitrequest` until the transfer completes. You can also gain throughput when there are some cycles of overhead before a read response.

Where reads are not pipelined, the throughput is reduced. When both the master and slave interfaces support pipelined read transfers, data flows in a continuous stream after the initial latency. You can use a pipelined read master that stores data in a FIFO to implement a custom DMA, hardware accelerator, or off-chip communication interface.

Figure 275. Pipelined Read Master



This example shows a pipelined read master that stores data in a FIFO. The master performs word accesses that are word-aligned and reads from sequential memory addresses. The transfer length is a multiple of the word size.

When the `go` bit is asserted, the master registers the `start_address` and `transfer_length` signals. The master begins issuing reads continuously on the next clock cycle until the length register reaches zero. In this example, the word size is four bytes so that the address always increments by four, and the length decrements by four. The `read` signal remains asserted unless the FIFO fills to a predetermined level. The address register increments and the length register decrements if the length has not reached 0 and a read is posted.

The master posts a read transfer every time the `read` signal is asserted and the `waitrequest` is deasserted. The master issues reads until the entire buffer has been read or `waitrequest` is asserted. An optional tracking block monitors the `done` bit. When the length register reaches zero, some reads are outstanding. The tracking logic prevents assertion of `done` until the last read completes, and monitors the number of reads posted to the interconnect so that it does not exceed the space remaining in the `readdata` FIFO. This example includes a counter that verifies that the following conditions are met:

- If a read is posted and `readdatavalid` is deasserted, the counter increments.
- If a read is not posted and `readdatavalid` is asserted, the counter decrements.

When the `length` register and the tracking logic counter reach zero, all the reads have completed and the `done` bit is asserted. The `done` bit is important if a second master overwrites the memory locations that the pipelined read master accesses. This bit guarantees that the reads have completed before the original data is overwritten.

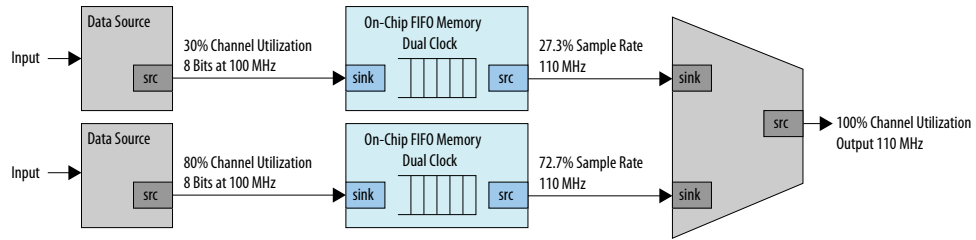
## 12.10.2 Multiplexer Examples

You can combine adapters with streaming components to create datapaths whose input and output streams have different properties. The following examples demonstrate datapaths in which the output stream exhibits higher performance than the input stream.

The diagram below illustrates a datapath that uses the dual clock version of the on-chip FIFO memory to boost the frequency of input data from 100 MHz to 110 MHz by sampling two input streams at differential rates. The on-chip FIFO memory has an input clock frequency of 100 MHz, and an output clock frequency of 110 MHz. The channel multiplexer runs at 110 MHz and samples one input stream 27.3 percent of the time, and the second 72.7 percent of the time. You must know what the typical and maximum input channel utilizations are before for this type of design. For example, if the first channel hits 50% utilization, the output stream exceeds 100% utilization.

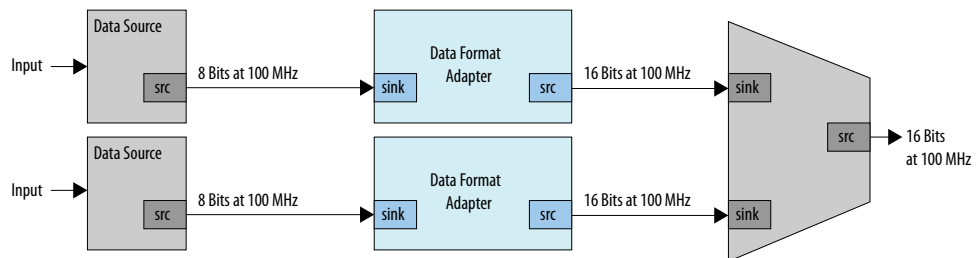


**Figure 276. Datapath that Doubles the Clock Frequency**



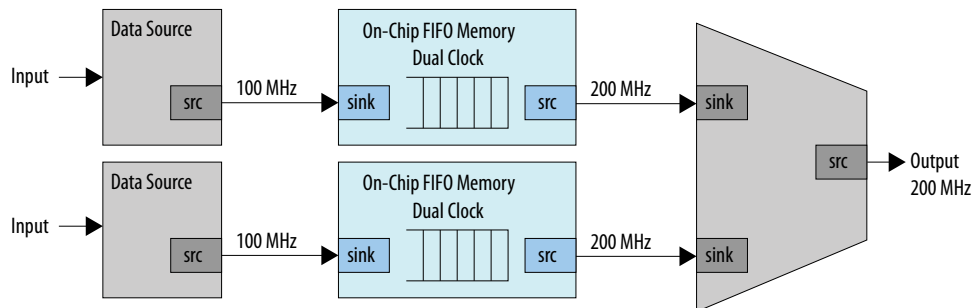
The diagram below illustrates a datapath that uses a data format adapter and Avalon-ST channel multiplexer to merge the 8-bit 100 MHz input from two streaming data sources into a single 16-bit 100 MHz streaming output. This example shows an output with double the throughput of each interface with a corresponding doubling of the data width.

**Figure 277. Datapath to Double Data Width and Maintain Original Frequency**



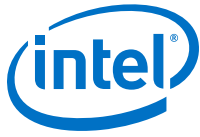
The diagram below illustrates a datapath that uses the dual clock version of the on-chip FIFO memory and Avalon-ST channel multiplexer to merge the 100 MHz input from two streaming data sources into a single 200 MHz streaming output. This example shows an output with double the throughput of each interface with a corresponding doubling of the clock frequency.

**Figure 278. Datapath to Boost the Clock Frequency**



## 12.11 Document Revision History

The table below indicates edits made to the *Optimizing Platform Designer System Performance* content since its creation.

**Table 189. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"><li>Changed instances of <i>Qsys Pro</i> to Platform Designer</li></ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"><li>Implemented Intel rebranding.</li><li>Implemented Platform Designer rebranding.</li></ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"><li>Added: <i>Reset Polarity and Synchronization in Platform Designer</i>.</li><li>Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i>.</li></ul>
2015.05.04	15.0.0	<i>Multiplexer Examples</i> , rearranged description text for the figures.
May 2013	13.0.0	AMBA APB support.
November 2012	12.1.0	AMBA AXI4 support.
June 2012	12.0.0	AMBA AXI3 support.
November 2011	11.1.0	New document release.

**Related Links**[Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 13 Component Interface Tcl Reference

---

Tcl commands allow you to perform a wide range of functions in Platform Designer. Command descriptions contain the Platform Designer phases where you can use the command, for example, main program, elaboration, composition, or fileset callback. This reference denotes optional command arguments in brackets [ ].

**Note:** Intel now refers to Qsys Pro as Platform Designer.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

For more information about procedures for creating IP component `_hw.tcl` files in the Platform Designer Component Editor, and supported interface standards, refer to *Creating Platform Designer Components* and *Platform Designer Interconnect*.

If you are developing an IP component to work with the Nios II processor, refer to *Publishing Component Information to Embedded Software* in section 3 of the *Nios II Software Developer's Handbook*, which describes how to publish hardware IP component information for embedded software tools, such as a C compiler and a Board Support Package (BSP) generator.

### Related Links

- [Avalon Interface Specifications](#)
- [AMBA Protocol Specifications](#)
- [Creating Platform Designer Components](#) on page 608
- [Platform Designer Interconnect](#) on page 659
- [Publishing Component Information to Embedded Software](#)  
In *Nios II Gen2 Software Developer's Handbook*

### 13.1 Platform Designer `_hw.tcl` Command Reference



### **13.1.1 Interfaces and Ports**

- [add\\_interface](#) on page 793
- [add\\_interface\\_port](#) on page 795
- [get\\_interfaces](#) on page 797
- [get\\_interface\\_assignment](#) on page 798
- [get\\_interface\\_assignments](#) on page 799
- [get\\_interface\\_ports](#) on page 800
- [get\\_interface\\_properties](#) on page 801
- [get\\_interface\\_property](#) on page 802
- [get\\_port\\_properties](#) on page 803
- [get\\_port\\_property](#) on page 804
- [set\\_interface\\_assignment](#) on page 805
- [set\\_interface\\_property](#) on page 807
- [set\\_port\\_property](#) on page 808
- [set\\_interface\\_upgrade\\_map](#) on page 809





### 13.1.1.1 add\_interface

#### Description

Adds an interface to your module. An interface represents a collection of related signals that are managed together in the parent system. These signals are implemented in the IP component's HDL, or exported from an interface from a child instance. As the IP component author, you choose the name of the interface.

#### Availability

Discovery, Main Program, Elaboration, Composition

#### Usage

```
add_interface <name> <type> <direction> [<associated_clock>]
```

#### Returns

No returns value.

#### Arguments

*name* A name you choose to identify an interface.

*type* The type of interface.

*direction* The interface direction.

*associated\_clock* (optional) (deprecated) For interfaces requiring associated clocks, use:  
`set_interface_property <interface> associatedClock <clockInterface>` For interfaces requiring associated resets, use: `set_interface_property <interface> associatedReset <resetInterface>`

#### Example

```
add_interface mm_slave avalon slave
add_interface my_export conduit end
set_interface_property my_export EXPORT_OF uart_0.external_connection
```

#### Notes

By default, interfaces are enabled. You can set the interface property `ENABLED` to `false` to disable an interface. If an interface is disabled, it is hidden and its ports are automatically terminated to their default values. Active high signals are terminated to 0. Active low signals are terminated to 1.

If the IP component is composed of child instances, the top-level interface is associated with a child instance's interface with `set_interface_property interface EXPORT_OF child_instance.interface`.

The following direction rules apply to Platform Designer-supported interfaces.



Interface Type	Direction
avalon	master, slave
axi	master, slave
tristate_conduit	master, slave
avalon_streaming	source, sink
interrupt	sender, receiver
conduit	end
clock	source, sink
reset	source, sink
nios_custom_instruction	slave

**Related Links**

- [add\\_interface\\_port](#) on page 795
- [get\\_interface\\_assignments](#) on page 799
- [get\\_interface\\_properties](#) on page 801
- [get\\_interfaces](#) on page 797



### 13.1.1.2 add\_interface\_port

#### Description

Adds a port to an interface on your module. The name must match the name of a signal on the top-level module in the HDL of your IP component. The port width and direction must be set before the end of the elaboration phase. You can set the port width as follows:

- In the Main program, you can set the port width to a fixed value or a width expression.
- If the port width is set to a fixed value in the Main program, you can update the width in the elaboration callback.

#### Availability

Main Program, Elaboration

#### Usage

```
add_interface_port <interface> <port> [<signal_type> <direction>  
<width_expression>]
```

#### Returns

#### Arguments

*interface* The name of the interface to which this port belongs.

*port* The name of the port. This name must match a signal in your top-level HDL for this IP component.

*signal\_type (optional)* The type of signal for this port, which must be unique. Refer to the *Avalon Interface Specifications* for the signal types available for each interface type.

*direction (optional)* The direction of the signal. Refer to *Direction Properties*.

*width\_expression (optional)* The width of the port, in bits. The width may be a fixed value, or a simple arithmetic expression of parameter values.

#### Example

```
fixed width:  
add_interface_port mm_slave s0_rdata readdata output 32  
  
width expression:  
add_parameter DATA_WIDTH INTEGER 32  
add_interface_port s0 rdata readdata output "DATA_WIDTH/2"
```

#### Related Links

- [add\\_interface](#) on page 793
- [get\\_port\\_properties](#) on page 803



- [get\\_port\\_property](#) on page 804
- [get\\_port\\_property](#) on page 804
- [Direction Properties](#) on page 893
- [Avalon Interface Specifications](#)



### 13.1.1.3 get\_interfaces

#### Description

Returns a list of top-level interfaces.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_interfaces
```

#### Returns

A list of the top-level interfaces exported from the system.

#### Arguments

No arguments.

#### Example

```
get_interfaces
```

#### Related Links

[add\\_interface](#) on page 793



#### 13.1.1.4 get\_interface\_assignment

##### Description

Returns the value of the specified assignment for the specified interface

##### Availability

Main Program, Elaboration, Validation, Composition

##### Usage

```
get_interface_assignment <interface> <assignment>
```

##### Returns

The value of the assignment.

##### Arguments

*interface* The name of a top-level interface.

*assignment* The name of an assignment.

##### Example

```
get_interface_assignment s1 embeddedsw.configuration.isFlash
```

##### Related Links

- [add\\_interface](#) on page 793
- [get\\_interface\\_assignments](#) on page 799
- [get\\_interfaces](#) on page 797



### 13.1.1.5 get\_interface\_assignments

#### Description

Returns the value of all interface assignments for the specified interface.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
get_interface_assignments <interface>
```

#### Returns

A list of assignment keys.

#### Arguments

*interface* The name of the top-level interface whose assignment is being retrieved.

#### Example

```
get_interface_assignments s1
```

#### Related Links

- [add\\_interface](#) on page 793
- [get\\_interface\\_assignment](#) on page 798
- [get\\_interfaces](#) on page 797



### 13.1.1.6 get\_interface\_ports

#### Description

Returns the names of all of the ports that have been added to a given interface. If the interface name is omitted, all ports for all interfaces are returned.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_interface_ports [<interface>]
```

#### Returns

A list of port names.

#### Arguments

*interface (optional)* The name of a top-level interface.

#### Example

```
get_interface_ports mm_slave
```

#### Related Links

- [add\\_interface\\_port](#) on page 795
- [get\\_port\\_property](#) on page 804
- [set\\_port\\_property](#) on page 808





### 13.1.1.7 get\_interface\_properties

#### Description

Returns the names of all the interface properties for the specified interface as a space separated list

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_interface_properties <interface>
```

#### Returns

A list of properties for the interface.

#### Arguments

*interface* The name of an interface.

#### Example

```
get_interface_properties interface
```

#### Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

#### Related Links

- [get\\_interface\\_property](#) on page 802
- [set\\_interface\\_property](#) on page 807
- [Avalon Interface Specifications](#)



### 13.1.1.8 get\_interface\_property

#### Description

Returns the value of a single interface property from the specified interface.

#### Availability

Discovery, Main Program, Elaboration, Composition, Fileset Generation

#### Usage

```
get_interface_property <interface> <property>
```

#### Returns

#### Arguments

*interface* The name of an interface.

*property* The name of the property whose value you want to retrieve. Refer to *Interface Properties*.

#### Example

```
get_interface_property mm_slave linewidthBursts
```

#### Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

#### Related Links

- [get\\_interface\\_properties](#) on page 801
- [set\\_interface\\_property](#) on page 807
- [Avalon Interface Specifications](#)



### 13.1.1.9 get\_port\_properties

#### Description

Returns a list of port properties.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_port_properties
```

#### Returns

A list of port properties. Refer to *Port Properties*.

#### Arguments

No arguments.

#### Example

```
get_port_properties
```

#### Related Links

- [add\\_interface\\_port](#) on page 795
- [get\\_port\\_property](#) on page 804
- [set\\_port\\_property](#) on page 808
- [Port Properties](#) on page 891



### 13.1.1.10 get\_port\_property

#### Description

Returns the value of a property for the specified port.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_port_property <port> <property>
```

#### Returns

The value of the property.

#### Arguments

*port* The name of the port.

*property* The name of a port property. Refer to *Port Properties*.

#### Example

```
get_port_property rdata WIDTH_VALUE
```

#### Related Links

- [add\\_interface\\_port](#) on page 795
- [get\\_port\\_properties](#) on page 803
- [set\\_port\\_property](#) on page 808
- [Port Properties](#) on page 891



### 13.1.1.11 set\_interface\_assignment

#### Description

Sets the value of the specified assignment for the specified interface.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
set_interface_assignment <interface> <assignment> [<value>]
```

#### Returns

No return value.

#### Arguments

*interface* The name of the top-level interface whose assignment is being set.

*assignment* The assignment whose value is being set.

*value (optional)* The new assignment value.

#### Example

```
set_interface_assignment s1 embeddedsw.configuration.isFlash 1
```

#### Notes

##### Assignments for Nios II Software Build Tools

Interface assignments provide extra data for the Nios II Software Build Tools working with the generated system.

##### Assignments for Platform Designer Tools

There are several assignments that guide behavior in the Platform Designer tools.

*qsys.ui.export\_name:* If present, this interface should always be exported when an instance is added to a Platform Designer system. The value is the requested name of the exported interface in the parent system.

*qsys.ui.connect:* If present, this interface should be auto-connected when an instance is added to a Platform Designer system. The value is a comma-separated list of other interfaces on the same instance that should be connected with this interface.



*ui.blockdiagram.direction*: If present, the direction of this interface in the block diagram is set by the user. The value is either "output" or "input".

#### Related Links

- [add\\_interface](#) on page 793
- [get\\_interface\\_assignment](#) on page 798
- [get\\_interface\\_assignments](#) on page 799



### 13.1.1.12 set\_interface\_property

#### Description

Sets the value of a property on an exported top-level interface. You can use this command to set the `EXPORT_OF` property to specify which interface of a child instance is exported via this top-level interface.

#### Availability

Main Program, Elaboration, Composition

#### Usage

```
set_interface_property <interface> <property> <value>
```

#### Returns

No return value.

#### Arguments

*interface* The name of an exported top-level interface.

*property* The name of the property Refer to *Interface Properties*.

*value* The new property value.

#### Example

```
set_interface_property clk_out EXPORT_OF clk.clk_out  
set_interface_property mm_slave linewidthBursts false
```

#### Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

#### Related Links

- [get\\_interface\\_properties](#) on page 801
- [get\\_interface\\_property](#) on page 802
- [Interface Properties](#) on page 885
- [Avalon Interface Specifications](#)



### 13.1.1.13 set\_port\_property

#### Description

Sets a port property.

#### Availability

Main Program, Elaboration

#### Usage

```
set_port_property <port> <property> [<value>]
```

#### Returns

The new value.

#### Arguments

*port* The name of the port.

*property* One of the supported properties. Refer to *Port Properties*.

*value (optional)* The value to set.

#### Example

```
set_port_property rdata WIDTH 32
```

#### Related Links

- [add\\_interface\\_port](#) on page 795
- [get\\_port\\_properties](#) on page 803
- [set\\_port\\_property](#) on page 808





### 13.1.1.14 set\_interface\_upgrade\_map

#### Description

Maps the interface name of an older version of an IP core to the interface name of the current IP core. The interface type must be the same between the older and newer versions of the IP cores. This allows system connections and properties to maintain proper functionality. By default, if the older and newer versions of IP core have the same name and type, then Platform Designer maintains all properties and connections automatically.

#### Availability

Parameter Upgrade

#### Usage

```
set_interface_upgrade_map { <old_interface_name> <new_interface_name>
<old_interface_name_2> <new_interface_name_2> ... }
```

#### Returns

No return value.

#### Arguments

{ <old_interface_name> <new_interface_name>}	List of mappings between between names of older and newer interfaces.
-------------------------------------------------	--------------------------------------------------------------------------

#### Example

```
set_interface_upgrade_map { avalon_master_interface
new_avalon_master_interface }
```



## 13.1.2 Parameters

- [add\\_parameter](#) on page 811
- [get\\_parameters](#) on page 812
- [get\\_parameter\\_properties](#) on page 813
- [get\\_parameter\\_property](#) on page 814
- [get\\_parameter\\_value](#) on page 815
- [get\\_string](#) on page 816
- [load\\_strings](#) on page 817
- [set\\_parameter\\_property](#) on page 818
- [set\\_parameter\\_value](#) on page 819
- [decode\\_address\\_map](#) on page 820



### 13.1.2.1 add\_parameter

#### Description

Adds a parameter to your IP component.

#### Availability

Main Program

#### Usage

```
add_parameter <name> <type> [<default_value> <description>]
```

#### Returns

#### Arguments

*name* The name of the parameter.

*type* The data type of the parameter Refer to *Parameter Type Properties*.

*default\_value (optional)* The initial value of the parameter in a new instance of the IP component.

*description (optional)* Explains the use of the parameter.

#### Example

```
add_parameter seed INTEGER 17 "The seed to use for data generation."
```

#### Notes

Most parameter types have a single GUI element for editing the parameter value. `string_list` and `integer_list` parameters are different, because they are edited as tables. A multi-column table can be created by grouping multiple into a single table. To edit multiple list parameters in a single table, the display items for the parameters must be added to a group with a `TABLE` hint:

```
add_parameter coefficients INTEGER_LIST add_parameter positions  
INTEGER_LIST add_display_item "" "Table Group" GROUP TABLE  
add_display_item "Table Group" coefficients PARAMETER  
add_display_item "Table Group" positions PARAMETER
```

#### Related Links

- [get\\_parameter\\_properties](#) on page 813
- [get\\_parameter\\_property](#) on page 814
- [get\\_parameter\\_value](#) on page 815
- [set\\_parameter\\_property](#) on page 818
- [set\\_parameter\\_value](#) on page 819
- [Parameter Type Properties](#) on page 889



### 13.1.2.2 get\_parameters

#### Description

Returns the names of all the parameters in the IP component.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_parameters
```

#### Returns

A list of parameter names

#### Arguments

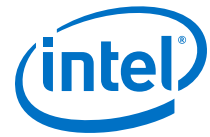
No arguments.

#### Example

```
get_parameters
```

#### Related Links

- [add\\_parameter](#) on page 811
- [get\\_parameter\\_property](#) on page 814
- [get\\_parameter\\_value](#) on page 815
- [get\\_parameters](#) on page 812
- [set\\_parameter\\_property](#) on page 818



### 13.1.2.3 get\_parameter\_properties

#### Description

Returns a list of all the parameter properties as a list of strings. The `get_parameter_property` and `set_parameter_property` commands are used to get and set the values of these properties, respectively.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_parameter_properties
```

#### Returns

A list of parameter property names. Refer to *Parameter Properties*.

#### Arguments

No arguments.

#### Example

```
set property_summary [ get_parameter_properties ]
```

#### Related Links

- [add\\_parameter](#) on page 811
- [get\\_parameter\\_property](#) on page 814
- [get\\_parameter\\_value](#) on page 815
- [get\\_parameters](#) on page 812
- [set\\_parameter\\_property](#) on page 818
- [Parameter Properties](#) on page 887



### 13.1.2.4 get\_parameter\_property

#### Description

Returns the value of a property of a parameter.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_parameter_property <parameter> <property>
```

#### Returns

The value of the property.

#### Arguments

*parameter* The name of the parameter whose property value is being retrieved.

*property* The name of the property. Refer to *Parameter Properties*.

#### Example

```
set enabled [ get_parameter_property parameter1 ENABLED ]
```

#### Related Links

- [add\\_parameter](#) on page 811
- [get\\_parameter\\_properties](#) on page 813
- [get\\_parameter\\_value](#) on page 815
- [get\\_parameters](#) on page 812
- [set\\_parameter\\_property](#) on page 818
- [set\\_parameter\\_value](#) on page 819
- [Parameter Properties](#) on page 887



### 13.1.2.5 get\_parameter\_value

#### Description

Returns the current value of a parameter defined previously with the `add_parameter` command.

#### Availability

Discovery, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_parameter_value <parameter>
```

#### Returns

The value of the parameter.

#### Arguments

*parameter* The name of the parameter whose value is being retrieved.

#### Example

```
set width [ get_parameter_value fifo_width ]
```

#### Notes

If `AFFECTS_ELABORATION` is `false` for a given parameter, `get_parameter_value` is not available for that parameter from the elaboration callback. If `AFFECTS_GENERATION` is `false` then it is not available from the generation callback.

#### Related Links

- [add\\_parameter](#) on page 811
- [get\\_parameter\\_property](#) on page 814
- [get\\_parameters](#) on page 812
- [set\\_parameter\\_property](#) on page 818
- [set\\_parameter\\_value](#) on page 819

### 13.1.2.6 get\_string

#### Description

Returns the value of an externalized string previously loaded by the `load_strings` command.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_string <identifier>
```

#### Returns

The externalized string.

#### Arguments

*identifier* The string identifier.

#### Example

```
hw.tcl:
load_strings test.properties
set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

test.properties:
DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

#### Notes

Use uppercase words separated with underscores to name string identifiers. If you are externalizing module properties, use the module property name for the string identifier:

```
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
```

If you are externalizing a parameter property, qualify the parameter property with the parameter name, with uppercase format, if needed:

```
set_parameter_property my_param DISPLAY_NAME [get_string
MY_PARAM_DISPLAY_NAME]
```

If you use a string to describe a string format, end the identifier with `_FORMAT`.

```
set formatted_string [ format [ get_string TWO_ARGUMENT_MESSAGE_FORMAT ]
"arg1" "arg2" ]
```

#### Related Links

[load\\_strings](#) on page 817





### 13.1.2.7 load\_strings

#### Description

Loads strings from an external `.properties` file.

#### Availability

Discovery, Main Program

#### Usage

`load_strings <path>`

#### Returns

No return value.

#### Arguments

*path* The path to the properties file.

#### Example

```
hw.tcl:
load_strings test.properties
set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

test.properties:
DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

#### Notes

Refer to the *Java Properties File* for properties file format. A `.properties` file is a text file with `KEY=value` pairs. For externalized strings, the `KEY` is a string identifier and the `value` is the externalized string.

For example:

```
TROGDOR = A dragon with a big beefy arm
```

#### Related Links

- [get\\_string](#) on page 816
- [Java Properties File](#)



### 13.1.2.8 set\_parameter\_property

#### Description

Sets a single parameter property.

#### Availability

Main Program, Edit, Elaboration, Validation, Composition

#### Usage

```
set_parameter_property <parameter> <property> <value>
```

#### Returns

#### Arguments

*parameter* The name of the parameter that is being set.

*property* The name of the property. Refer to *Parameter Properties*.

*value* The new value for the property.

#### Example

```
set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}
```

#### Related Links

- [add\\_parameter](#) on page 811
- [get\\_parameter\\_properties](#) on page 813
- [set\\_parameter\\_property](#) on page 818
- [Parameter Properties](#) on page 887



### 13.1.2.9 set\_parameter\_value

#### Description

Sets a parameter value. The value of a derived parameter can be updated by the IP component in the elaboration callback or the edit callback. Any changes to the value of a derived parameter in the edit callback is not preserved.

#### Availability

Edit, Elaboration, Validation, Composition, Parameter Upgrade

#### Usage

```
set_parameter_value <parameter> <value>
```

#### Returns

No return value.

#### Arguments

*parameter* The name of the parameter that is being set.

*value* Specifies the new parameter value.

#### Example

```
set_parameter_value half_clock_rate [ expr { [ get_parameter_value  
clock_rate ] / 2 } ]
```



### 13.1.2.10 decode\_address\_map

#### Description

Converts an XML-formatted address map into a list of Tcl lists. Each inner list is in the correct format for conversion to an array. The XML code that describes each slave includes: its name, start address, and end address.

#### Availability

Elaboration, Generation, Composition

#### Usage

```
decode_address_map <address_map_XML_string>
```

#### Returns

No return value.

#### Arguments

*address\_mapXML\_string* An XML string that describes the address map of a master.

#### Example

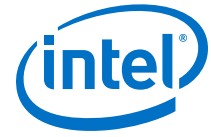
In this example, the code describes the address map for the master that accesses the `ext_ssram`, `sys_clk_timer` and `sysid` slaves. The format of the string may differ from the example below; it may have different white space between the elements and include additional attributes or elements. Use the `decode_address_map` command to decode the code that represents a master's address map to ensure that your code works with future versions of the address map.

```
<address-map>
  <slave name='ext_ssram' start='0x01000000' end='0x01200000' />
  <slave name='sys_clk_timer' start='0x02120800' end='0x02120820' />
  <slave name='sysid' start='0x021208B8' end='0x021208C0' />
</address-map>
```

#### Note:

Intel recommends that you use the code provided below to enumerate over the IP components within an address map, rather than writing your own parser.

```
set address_map_xml [get_parameter_value my_map_param]
set address_map_dec [decode_address_map $address_map_xml]
foreach i $address_map_dec {
  array set info $i
  send_message info "Connected to slave $info(name)"
}
```



### **13.1.3 Display Items**

- [add\\_display\\_item](#) on page 822
- [get\\_display\\_items](#) on page 824
- [get\\_display\\_item\\_properties](#) on page 825
- [get\\_display\\_item\\_property](#) on page 826
- [set\\_display\\_item\\_property](#) on page 827



### 13.1.3.1 add\_display\_item

#### Description

Specifies the following aspects of the IP component display:

- Creates logical groups for an IP component's parameters. For example, to create separate groups for the IP component's timing, size, and simulation parameters. An IP component displays the groups and parameters in the order that you specify the display items in the `_hw.tcl` file.
- Groups a list of parameters to create multi-column tables.
- Specifies an image to provide representation of a parameter or parameter group.
- Creates a button by adding a display item of type `action`. The display item includes the name of the callback to run.

#### Availability

Main Program

#### Usage

```
add_display_item <parent_group> <id> <type> [<args>]
```

#### Returns

#### Arguments

*parent\_group* Specifies the group to which a display item belongs

*id* The identifier for the display item. If the item being added is a parameter, this is the parameter name. If the item is a group, this is the group name.

*type* The type of the display item. Refer to *Display Item Kind Properties*.

*args (optional)* Provides extra information required for display items.

#### Example

```
add_display_item "Timing" read_latency PARAMETER
add_display_item "Sounds" speaker_image_id ICON speaker.jpg
```



## Notes

The following examples illustrate further illustrate the use of arguments:

- `add_display_item groupName id icon path-to-image-file`
- `add_display_item groupName parameterName parameter`
- `add_display_item groupName id text "your-text"`

The your-text argument is a block of text that is displayed in the GUI. Some simple HTML formatting is allowed, such as `<b>` and `<i>`, if the text starts with `<html>`.

- `add_display_item parentGroupName childGroupName group [tab]`  
The tab is an optional parameter. If present, the group appears in separate tab in the GUI for the instance.
- `add_display_item parentGroupName actionName action  
buttonClickCallbackProc`

## Related Links

- [get\\_display\\_item\\_properties](#) on page 825
- [get\\_display\\_item\\_property](#) on page 826
- [get\\_display\\_items](#) on page 824
- [set\\_display\\_item\\_property](#) on page 827
- [Display Item Kind Properties](#) on page 895



### 13.1.3.2 get\_display\_items

#### Description

Returns a list of all items to be displayed as part of the parameterization GUI.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_display_items
```

#### Returns

List of display item IDs.

#### Arguments

No arguments.

#### Example

```
get_display_items
```

#### Related Links

- [add\\_display\\_item](#) on page 822
- [get\\_display\\_item\\_properties](#) on page 825
- [get\\_display\\_item\\_property](#) on page 826
- [set\\_display\\_item\\_property](#) on page 827





### 13.1.3.3 get\_display\_item\_properties

#### Description

Returns a list of names of the properties of display items that are part of the parameterization GUI.

#### Availability

Main Program

#### Usage

```
get_display_item_properties
```

#### Returns

A list of display item property names. Refer to *Display Item Properties*.

#### Arguments

No arguments.

#### Example

```
get_display_item_properties
```

#### Related Links

- [add\\_display\\_item](#) on page 822
- [get\\_display\\_item\\_property](#) on page 826
- [set\\_display\\_item\\_property](#) on page 827
- [Display Item Properties](#) on page 894



### 13.1.3.4 get\_display\_item\_property

#### Description

Returns the value of a specific property of a display item that is part of the parameterization GUI.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
get_display_item_property <display_item> <property>
```

#### Returns

The value of a display item property.

#### Arguments

*display\_item* The id of the display item.

*property* The name of the property. Refer to *Display Item Properties*.

#### Example

```
set my_label [get_display_item_property my_action DISPLAY_NAME]
```

#### Related Links

- [add\\_display\\_item](#) on page 822
- [get\\_display\\_item\\_properties](#) on page 825
- [get\\_display\\_items](#) on page 824
- [set\\_display\\_item\\_property](#) on page 827
- [Display Item Properties](#) on page 894



### 13.1.3.5 set\_display\_item\_property

#### Description

Sets the value of specific property of a display item that is part of the parameterization GUI.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition

#### Usage

```
set_display_item_property <display_item> <property> <value>
```

#### Returns

No return value.

#### Arguments

*display\_item* The name of the display item whose property value is being set.

*property* The property that is being set. Refer to *Display Item Properties*.

*value* The value to set.

#### Example

```
set_display_item_property my_action DISPLAY_NAME "Click Me"  
set_display_item_property my_action DESCRIPTION "clicking this button runs  
the click_me_callback proc in the hw.tcl file"
```

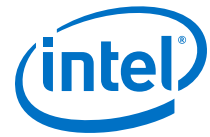
#### Related Links

- [add\\_display\\_item](#) on page 822
- [get\\_display\\_item\\_properties](#) on page 825
- [get\\_display\\_item\\_property](#) on page 826
- [Display Item Properties](#) on page 894



### **13.1.4 Module Definition**

[add\\_documentation\\_link](#) on page 829  
[get\\_module\\_assignment](#) on page 830  
[get\\_module\\_assignments](#) on page 831  
[get\\_module\\_ports](#) on page 832  
[get\\_module\\_properties](#) on page 833  
[get\\_module\\_property](#) on page 834  
[send\\_message](#) on page 835  
[set\\_module\\_assignment](#) on page 836  
[set\\_module\\_property](#) on page 837  
[add\\_hdl\\_instance](#) on page 838  
[package](#) on page 839



### 13.1.4.1 add\_documentation\_link

#### Description

Allows you to link to documentation for your IP component.

#### Availability

Discovery, Main Program

#### Usage

```
add_documentation_link <title> <path>
```

#### Returns

No return value.

#### Arguments

*title* The title of the document for use on menus and buttons.

*path* A path to the IP component documentation, using a syntax that provides the entire URL, not a relative path. For example: `http://www.mydomain.com/my_memory_controller.html` or `file:///datasheet.txt`

#### Example

```
add_documentation_link "Avalon Verification IP Suite User Guide" http://  
www.altera.com/literature/ug/ug_avalon_verification_ip.pdf
```



### 13.1.4.2 get\_module\_assignment

#### Description

This command returns the value of an assignment. You can use the `get_module_assignment` and `set_module_assignment` and the `get_interface_assignment` and `set_interface_assignment` commands to provide information about the IP component to embedded software tools and applications.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
get_module_assignment <assignment>
```

#### Returns

The value of the assignment

#### Arguments

*assignment* The name of the assignment whose value is being retrieved

#### Example

```
get_module_assignment embeddedsw.CMacro.colorSpace
```

#### Related Links

- [get\\_module\\_assignments](#) on page 831
- [set\\_module\\_assignment](#) on page 836



### 13.1.4.3 get\_module\_assignments

#### Description

Returns the names of the module assignments.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
get_module_assignments
```

#### Returns

A list of assignment names.

#### Arguments

No arguments.

#### Example

```
get_module_assignments
```

#### Related Links

- [get\\_module\\_assignment](#) on page 830
- [set\\_module\\_assignment](#) on page 836



#### 13.1.4.4 get\_module\_ports

##### Description

Returns a list of the names of all the ports which are currently defined.

##### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

##### Usage

```
get_module_ports
```

##### Returns

A list of port names.

##### Arguments

No arguments.

##### Example

```
get_module_ports
```

##### Related Links

- [add\\_interface](#) on page 793
- [add\\_interface\\_port](#) on page 795





### 13.1.4.5 get\_module\_properties

#### Description

Returns the names of all the module properties as a list of strings. You can use the `get_module_property` and `set_module_property` commands to get and set values of individual properties. The value returned by this command is always the same for a particular version of Platform Designer

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_module_properties
```

#### Returns

List of strings. Refer to *Module Properties*.

#### Arguments

No arguments.

#### Example

```
get_module_properties
```

#### Related Links

- [get\\_module\\_property](#) on page 834
- [set\\_module\\_property](#) on page 837
- [Module Properties](#) on page 897



### 13.1.4.6 get\_module\_property

#### Description

Returns the value of a single module property.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_module_property <property>
```

#### Returns

Various.

#### Arguments

*property* The name of the property, Refer to *Module Properties*.

#### Example

```
set my_name [ get_module_property NAME ]
```

#### Related Links

- [get\\_module\\_properties](#) on page 833
- [set\\_module\\_property](#) on page 837
- [Module Properties](#) on page 897



### 13.1.4.7 send\_message

#### Description

Sends a message to the user of the IP component. The message text is normally interpreted as HTML. You can use the <b> element to provide emphasis. If you do not want the message text to be interpreted as HTML, then pass a list as the message level, for example, { Info Text }.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
send_message <level> <message>
```

#### Returns

No return value .

#### Arguments

*level* The following message levels are supported:

- **ERROR**--Provides an error message. The Platform Designer system cannot be generated with existing error messages.
- **WARNING**--Provides a warning message.
- **INFO**--Provides an informational message.
- **PROGRESS**--Reports progress during generation.
- **DEBUG**--Provides a debug message when debug mode is enabled.

*message* The text of the message.

#### Example

```
send_message ERROR "The system is down!"  
send_message { Info Text } "The system is up!"
```



### 13.1.4.8 set\_module\_assignment

#### Description

Sets the value of the specified assignment.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
set_module_assignment <assignment> [<value>]
```

#### Returns

No return value.

#### Arguments

*assignment* The assignment whose value is being set

*value (optional)* The value of the assignment

#### Example

```
set_module_assignment embeddedsw.CMacro.colorSpace CMYK
```

#### Related Links

- [get\\_module\\_assignment](#) on page 830
- [get\\_module\\_assignments](#) on page 831



### 13.1.4.9 set\_module\_property

#### Description

Allows you to set the values for module properties.

#### Availability

Discovery, Main Program

#### Usage

```
set_module_property <property> <value>
```

#### Returns

No return value.

#### Arguments

*property* The name of the property. Refer to *Module Properties*.

*value* The new value of the property.

#### Example

```
set_module_property VERSION 10.0
```

#### Related Links

- [get\\_module\\_properties](#) on page 833
- [get\\_module\\_property](#) on page 834
- [Module Properties](#) on page 897



### 13.1.4.10 add\_hdl\_instance

#### Description

Adds an instance of a predefined module, referred to as a *child* or *child instance*. The HDL entity generated from this instance can be instantiated and connected within this IP component's HDL.

#### Availability

Main Program, Elaboration, Composition

#### Usage

```
add_hdl_instance <entity_name> <ip_core_type> [<version>]
```

#### Returns

The entity name of the added instance.

#### Arguments

*entity\_name* Specifies a unique local name that you can use to manipulate the instance. This name is used in the generated HDL to identify the instance.

*ip\_core\_type* The type refers to a kind of instance available in the IP Catalog, for example `altera_avalon_uart`.

*version (optional)* The required version of the specified instance type. If no version is specified, the latest version is used.

#### Example

```
add_hdl_instance my_uart altera_avalon_uart
```

#### Related Links

- [get\\_instance\\_parameter\\_value](#) on page 856
- [get\\_instance\\_parameters](#) on page 854
- [get\\_instances](#) on page 846
- [set\\_instance\\_parameter\\_value](#) on page 859



### 13.1.4.11 package

#### Description

Allows you to specify a particular version of the Platform Designer software to avoid software compatibility issues, and to determine which version of the `_hw.tcl` API to use for the IP component. You must use the `package` command at the beginning of your `_hw.tcl` file.

#### Availability

Main Program

#### Usage

```
package require -exact qsys <version>
```

#### Returns

No return value

#### Arguments

*version* The version of Platform Designer that you require, such as 14.1.

#### Example

```
package require -exact qsys 14.1
```



### **13.1.5 Composition**

[add\\_instance](#) on page 841  
[add\\_connection](#) on page 842  
[get\\_connections](#) on page 843  
[get\\_connection\\_parameters](#) on page 844  
[get\\_connection\\_parameter\\_value](#) on page 845  
[get\\_instances](#) on page 846  
[get\\_instance\\_interfaces](#) on page 847  
[get\\_instance\\_interface\\_ports](#) on page 848  
[get\\_instance\\_interface\\_properties](#) on page 849  
[get\\_instance\\_property](#) on page 850  
[set\\_instance\\_property](#) on page 851  
[get\\_instance\\_properties](#) on page 852  
[get\\_instance\\_interface\\_property](#) on page 853  
[get\\_instance\\_parameters](#) on page 854  
[get\\_instance\\_parameter\\_property](#) on page 855  
[get\\_instance\\_parameter\\_value](#) on page 856  
[get\\_instance\\_port\\_property](#) on page 857  
[set\\_connection\\_parameter\\_value](#) on page 858  
[set\\_instance\\_parameter\\_value](#) on page 859





### 13.1.5.1 add\_instance

#### Description

Adds an instance of an IP component, referred to as a child or child instance to the subsystem. You can use this command to create IP components that are composed of other IP component instances. The HDL for this subsystem generates; There is no need to write custom HDL for the IP component.

#### Availability

Main Program, Composition

#### Usage

```
add_instance <name> <type> [<version>]
```

#### Returns

No return value.

#### Arguments

*name* Specifies a unique local name that you can use to manipulate the instance. This name is used in the generated HDL to identify the instance.

*type* The type refers to a type available in the IP Catalog, for example altera\_avalon\_uart.

*version (optional)* The required version of the specified type. If no version is specified, the highest available version is used.

#### Example

```
add_instance my_uart altera_avalon_uart  
add_instance my_uart altera_avalon_uart 14.1
```

#### Related Links

- [add\\_connection](#) on page 842
- [get\\_instance\\_interface\\_property](#) on page 853
- [get\\_instance\\_parameter\\_value](#) on page 856
- [get\\_instance\\_parameters](#) on page 854
- [get\\_instance\\_property](#) on page 850
- [get\\_instances](#) on page 846
- [set\\_instance\\_parameter\\_value](#) on page 859



### 13.1.5.2 add\_connection

#### Description

Connects the named interfaces on child instances together using an appropriate connection type. Both interface names consist of a child instance name, followed by the name of an interface provided by that module. For example, `mux0.out` is the interface named `out` on the instance named `mux0`. Be careful to connect the start to the end, and not the other way around.

#### Availability

Main Program, Composition

#### Usage

```
add_connection <start> [<end> <kind> <name>]
```

#### Returns

The name of the newly added connection in `start.point/end.point` format.

#### Arguments

*start* The start interface to be connected, in  
<instance\_name>.<interface\_name> format.

*end (optional)* The end interface to be connected,  
<instance\_name>.<interface\_name>.

*kind (optional)* The type of connection, such as `avalon` or `clock`.

*name (optional)* A custom name for the connection. If unspecified, the name will be  
<start\_instance>.<interface>.<end\_instance><interface>

#### Example

```
add_connection dma.read_master sdr.s1 avalon
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_interfaces](#) on page 847



### 13.1.5.3 get\_connections

#### Description

Returns a list of all connections in the composed subsystem.

#### Availability

Main Program, Composition

#### Usage

```
get_connections
```

#### Returns

A list of connections.

#### Arguments

No arguments.

#### Example

```
set all_connections [ get_connections ]
```

#### Related Links

[add\\_connection](#) on page 842



#### 13.1.5.4 get\_connection\_parameters

##### Description

Returns a list of parameters found on a connection.

##### Availability

Main Program, Composition

##### Usage

```
get_connection_parameters <connection>
```

##### Returns

A list of parameter names

##### Arguments

*connection* The connection to query.

##### Example

```
get_connection_parameters cpu.data_master/dma0.csr
```

##### Related Links

- [add\\_connection](#) on page 842
- [get\\_connection\\_parameter\\_value](#) on page 845



### 13.1.5.5 get\_connection\_parameter\_value

#### Description

Returns the value of a parameter on the connection. Parameters represent aspects of the connection that can be modified once the connection is created, such as the base address for an Avalon Memory Mapped connection.

#### Availability

Composition

#### Usage

```
get_connection_parameter_value <connection> <parameter>
```

#### Returns

The value of the parameter.

#### Arguments

*connection* The connection to query.

*parameter* The name of the parameter.

#### Example

```
get_connection_parameter_value cpu.data_master/dma0.csr baseAddress
```

#### Related Links

- [add\\_connection](#) on page 842
- [get\\_connection\\_parameters](#) on page 844



### 13.1.5.6 get\_instances

#### Description

Returns a list of the instance names for all child instances in the system.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
get_instances
```

#### Returns

A list of child instance names.

#### Arguments

No arguments.

#### Example

```
get_instances
```

#### Notes

This command can be used with instances created by either `add_instance` or `add_hdl_instance`.

#### Related Links

- [add\\_hdl\\_instance](#) on page 838
- [add\\_instance](#) on page 841
- [get\\_instance\\_parameter\\_value](#) on page 856
- [get\\_instance\\_parameters](#) on page 854
- [set\\_instance\\_parameter\\_value](#) on page 859



### 13.1.5.7 get\_instance\_interfaces

#### Description

Returns a list of interfaces found in a child instance. The list of interfaces can change if the parameterization of the instance changes.

#### Availability

Validation, Composition

#### Usage

```
get_instance_interfaces <instance>
```

#### Returns

A list of interface names.

#### Arguments

*instance* The name of the child instance.

#### Example

```
get_instance_interfaces pixel_converter
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_interface\\_ports](#) on page 848
- [get\\_instance\\_interfaces](#) on page 847



### 13.1.5.8 get\_instance\_interface\_ports

#### Description

Returns a list of ports found in an interface of a child instance.

#### Availability

Validation, Composition, Fileset Generation

#### Usage

```
get_instance_interface_ports <instance> <interface>
```

#### Returns

A list of port names found in the interface.

#### Arguments

*instance* The name of the child instance.

*interface* The name of an interface on the child instance.

#### Example

```
set port_names [ get_instance_interface_ports cpu data_master ]
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_interfaces](#) on page 847
- [get\\_instance\\_port\\_property](#) on page 857





### 13.1.5.9 get\_instance\_interface\_properties

#### Description

Returns the names of all of the properties of the specified interface

#### Availability

Validation, Composition

#### Usage

```
get_instance_interface_properties <instance> <interface>
```

#### Returns

List of property names.

#### Arguments

*instance* The name of the child instance.

*interface* The name of an interface on the instance.

#### Example

```
set properties [ get_instance_interface_properties cpu data_master ]
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_interface\\_property](#) on page 853
- [get\\_instance\\_interfaces](#) on page 847



### 13.1.5.10 get\_instance\_property

#### Description

Returns the value of a single instance property.

#### Availability

Main Program, Elaboration, Validation, Composition, Fileset Generation

#### Usage

```
get_instance_property <instance> <property>
```

#### Returns

Various.

#### Arguments

*instance* The name of the instance.

*property* The name of the property. Refer to *Instance Properties*.

#### Example

```
set my_name [ get_instance_property myinstance NAME ]
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_properties](#) on page 852
- [set\\_instance\\_property](#) on page 851
- [Instance Properties](#) on page 886



### 13.1.5.11 set\_instance\_property

#### Description

Allows a user to set the properties of a child instance.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
set_instance_property <instance> <property> <value>
```

#### Returns

#### Arguments

*instance* The name of the instance.

*property* The name of the property to set. Refer to *Instance Properties*.

*value* The new property value.

#### Example

```
set_instance_property myinstance SUPPRESS_ALL_WARNINGS true
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_properties](#) on page 852
- [get\\_instance\\_property](#) on page 850
- [Instance Properties](#) on page 886



### 13.1.5.12 get\_instance\_properties

#### Description

Returns the names of all the instance properties as a list of strings. You can use the `get_instance_property` and `set_instance_property` commands to get and set values of individual properties. The value returned by this command is always the same for a particular version of Platform Designer

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_instance_properties
```

#### Returns

List of strings. Refer to *Instance Properties*.

#### Arguments

No arguments.

#### Example

```
get_instance_properties
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_property](#) on page 850
- [set\\_instance\\_property](#) on page 851
- [Instance Properties](#) on page 886



### 13.1.5.13 get\_instance\_interface\_property

#### Description

Returns the value of a property for an interface in a child instance.

#### Availability

Validation, Composition

#### Usage

```
get_instance_interface_property <instance> <interface> <property>
```

#### Returns

The value of the property.

#### Arguments

*instance* The name of the child instance.

*interface* The name of an interface on the child instance.

*property* The name of the property of the interface.

#### Example

```
set value [ get_instance_interface_property cpu data_master setupTime ]
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_interfaces](#) on page 847

### 13.1.5.14 get\_instance\_parameters

#### Description

Returns a list of names of the parameters on a child instance that can be set using `set_instance_parameter_value`. It omits parameters that are derived and those that have the `SYSTEM_INFO` parameter property set.

#### Availability

Main Program, Elaboration, Validation, Composition

#### Usage

```
get_instance_parameters <instance>
```

#### Returns

A list of parameters in the instance.

#### Arguments

*instance* The name of the child instance.

#### Example

```
set parameters [ get_instance_parameters instance ]
```

#### Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

#### Related Links

- [add\\_hdl\\_instance](#) on page 838
- [add\\_instance](#) on page 841
- [get\\_instance\\_parameter\\_value](#) on page 856
- [get\\_instances](#) on page 846
- [set\\_instance\\_parameter\\_value](#) on page 859



### 13.1.5.15 get\_instance\_parameter\_property

#### Description

Returns the value of a property on a parameter in a child instance. Parameter properties are metadata that describe how the Platform Designer tools use the parameter.

#### Availability

Validation, Composition

#### Usage

```
get_instance_parameter_property <instance> <parameter> <property>
```

#### Returns

The value of the parameter property.

#### Arguments

*instance* The name of the child instance.

*parameter* The name of the parameter in the instance.

*property* The name of the property of the parameter. Refer to *Parameter Properties*.

#### Example

```
get_instance_parameter_property instance parameter property
```

#### Related Links

- [add\\_instance](#) on page 841
- [Parameter Properties](#) on page 887

### 13.1.5.16 get\_instance\_parameter\_value

#### Description

Returns the value of a parameter in a child instance. You cannot use this command to get the value of parameters whose values are derived or those that are defined using the SYSTEM\_INFO parameter property.

#### Availability

Elaboration, Validation, Composition

#### Usage

```
get_instance_parameter_value <instance> <parameter>
```

#### Returns

The value of the parameter.

#### Arguments

*instance* The name of the child instance.

*parameter* Specifies the parameter whose value is being retrieved.

#### Example

```
set dpi [ get_instance_parameter_value pixel_converter input_DPI ]
```

#### Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

#### Related Links

- [add\\_hdl\\_instance](#) on page 838
- [add\\_instance](#) on page 841
- [get\\_instance\\_parameters](#) on page 854
- [get\\_instances](#) on page 846
- [set\\_instance\\_parameter\\_value](#) on page 859





### 13.1.5.17 `get_instance_port_property`

#### Description

Returns the value of a property of a port contained by an interface in a child instance.

#### Availability

Validation, Composition, Fileset Generation

#### Usage

```
get_instance_port_property <instance> <port> <property>
```

#### Returns

The value of the property for the port.

#### Arguments

*instance* The name of the child instance.

*port* The name of a port in one of the interfaces on the child instance.

*property* The property whose value is being retrieved. Only the following port properties can be queried on ports of child instances: `ROLE`, `DIRECTION`, `WIDTH`, `WIDTH_EXPR` and `VHDL_TYPE`. Refer to *Port Properties*.

#### Example

```
get_instance_port_property instance port property
```

#### Related Links

- [add\\_instance](#) on page 841
- [get\\_instance\\_interface\\_ports](#) on page 848
- [Port Properties](#) on page 891



### 13.1.5.18 set\_connection\_parameter\_value

#### Description

Sets the value of a parameter of the connection. The start and end are each interface names of the format `<instance>.<interface>`. Connection parameters depend on the type of connection, for Avalon-MM they include base addresses and arbitration priorities.

#### Availability

Main Program, Composition

#### Usage

```
set_connection_parameter_value <connection> <parameter> <value>
```

#### Returns

No return value.

#### Arguments

*connection* Specifies the name of the connection as returned by the `add_connectioncommand`. It is of the form `start.point/end.point`.

*parameter* The name of the parameter.

*value* The new parameter value.

#### Example

```
set_connection_parameter_value cpu.data_master/dma0.csr baseAddress  
"0x000a0000"
```

#### Related Links

- [add\\_connection](#) on page 842
- [get\\_connection\\_parameter\\_value](#) on page 845



### 13.1.5.19 set\_instance\_parameter\_value

#### Description

Sets the value of a parameter for a child instance. Derived parameters and SYSTEM\_INFO parameters for the child instance can not be set with this command.

#### Availability

Main Program, Elaboration, Composition

#### Usage

```
set_instance_parameter_value <instance> <parameter> <value>
```

#### Returns

Vo return value.

#### Arguments

*instance* Specifies the name of the child instance.

*parameter* Specifies the parameter that is being set.

*value* Specifies the new parameter value.

#### Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

#### Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

#### Related Links

- [add\\_hdl\\_instance](#) on page 838
- [add\\_instance](#) on page 841
- [get\\_instance\\_parameter\\_value](#) on page 856
- [get\\_instances](#) on page 846



### 13.1.6 Fileset Generation

[add\\_fileset](#) on page 861  
[add\\_fileset\\_file](#) on page 862  
[set\\_fileset\\_property](#) on page 863  
[get\\_fileset\\_file\\_attribute](#) on page 864  
[set\\_fileset\\_file\\_attribute](#) on page 865  
[get\\_fileset\\_properties](#) on page 866  
[get\\_fileset\\_property](#) on page 867  
[get\\_fileset\\_sim\\_properties](#) on page 868  
[set\\_fileset\\_sim\\_properties](#) on page 869  
[create\\_temp\\_file](#) on page 870



### 13.1.6.1 add\_fileset

#### Description

Adds a generation fileset for a particular target as specified by the `kind`. Platform Designer calls the target (`SIM_VHDL`, `SIM_VERILOG`, `QUARTUS_SYNTH`, or `EXAMPLE_DESIGN`) when the specified generation target is requested. You can define multiple filesets for each kind of fileset. Platform Designer passes a single argument to the specified callback procedure. The value of the argument is a generated name, which you must use in the top-level module or entity declaration of your IP component. To override this generated name, you can set the fileset property `TOP_LEVEL`.

#### Availability

Main Program

#### Usage

```
add_fileset <name> <kind> [<callback_proc> <display_name>]
```

#### Returns

No return value.

#### Arguments

*name* The name of the fileset.

*kind* The kind of fileset. Refer to *Fileset Properties*.

*callback\_proc (optional)* A string identifying the name of the callback procedure. If you add files in the global section, you can then specify a blank callback procedure.

*display\_name (optional)* A display string to identify the fileset.

#### Example

```
add_fileset my_synthesis_fileset QUARTUS_SYNTH mySynthCallbackProc "My
Synthesis"
proc mySynthCallbackProc { topLevelName } { ... }
```

#### Notes

If using the `TOP_LEVEL` fileset property, all parameterizations of the component must use identical HDL.

#### Related Links

- [add\\_fileset\\_file](#) on page 862
- [get\\_fileset\\_property](#) on page 867
- [Fileset Properties](#) on page 899



### 13.1.6.2 add\_fileset\_file

#### Description

Adds a file to the generation directory. You can specify source file locations with either an absolute path, or a path relative to the IP component's `_hw.tcl` file. When you use the `add_fileset_file` command in a fileset callback, the Intel Quartus Prime software compiles the files in the order that they are added.

#### Availability

Main Program, Fileset Generation

#### Usage

```
add_fileset_file <output_file> <file_type> <file_source> <path_or_contents>
[<attributes>]
```

#### Returns

No return value.

#### Arguments

*output\_file* Specifies the location to store the file after Platform Designer generation

*file\_type* The kind of file. Refer to *File Kind Properties*.

*file\_source* Specifies whether the file is being added by path, or by file contents. Refer to *File Source Properties*.

*path\_or\_contents* When the *file\_source* is `PATH`, specifies the file to be copied to *output\_file*. When the *file\_source* is `TEXT`, specifies the text contents to be stored in the file.

*attributes* (optional) An optional list of file attributes. Typically used to specify that a file is intended for use only in a particular simulator. Refer to *File Attribute Properties*.

#### Example

```
add_fileset_file "./implementation/rx_pma.sv" SYSTEM_VERILOG PATH
synth_rx_pma.sv
add_fileset_file gui.sv SYSTEM_VERILOG TEXT "Customize your IP core"
```

#### Related Links

- [add\\_fileset](#) on page 861
- [get\\_fileset\\_file\\_attribute](#) on page 864
- [File Kind Properties](#) on page 903
- [File Source Properties](#) on page 904
- [File Attribute Properties](#) on page 902



### 13.1.6.3 set\_fileset\_property

#### Description

Allows you to set the properties of a fileset.

#### Availability

Main Program, Elaboration, Fileset Generation

#### Usage

```
set_fileset_property <fileset> <property> <value>
```

#### Returns

No return value.

#### Arguments

*fileset* The name of the fileset.

*property* The name of the property to set. Refer to *Fileset Properties*.

*value* The new property value.

#### Example

```
set_fileset_property mySynthFileset TOP_LEVEL simple_uart
```

#### Notes

When a fileset callback is called, the callback procedure will be passed a single argument. The value of this argument is a generated name which must be used in the top-level module or entity declaration of your IP component. If set, the `TOP_LEVEL` specifies a fixed name for the top-level name of your IP component.

The `TOP_LEVEL` property must be set in the global section. It cannot be set in a fileset callback.

If using the `TOP_LEVEL` fileset property, all parameterizations of the IP component must use identical HDL.

#### Related Links

- [add\\_fileset](#) on page 861
- [Fileset Properties](#) on page 899



### 13.1.6.4 get\_fileset\_file\_attribute

#### Description

Returns the attribute of a fileset file.

#### Availability

Main Program, Fileset Generation

#### Usage

```
get_fileset_file_attribute <output_file> <attribute>
```

#### Returns

Value of the fileset File attribute.

#### Arguments

*output\_file* Location of the output file.

*attribute* Specifies the name of the attribute Refer to *File Attribute Properties*.

#### Example

```
get_fileset_file_attribute my_file.sv ALDEC_SPECIFIC
```

#### Related Links

- [add\\_fileset](#) on page 861
- [add\\_fileset\\_file](#) on page 862
- [get\\_fileset\\_file\\_attribute](#) on page 864
- [File Attribute Properties](#) on page 902
- [add\\_fileset](#) on page 861
- [add\\_fileset\\_file](#) on page 862
- [get\\_fileset\\_file\\_attribute](#) on page 864
- [File Attribute Properties](#) on page 902





### 13.1.6.5 set\_fileset\_file\_attribute

#### Description

Sets the attribute of a fileset file.

#### Availability

Main Program, Fileset Generation

#### Usage

```
set_fileset_file_attribute <output_file> <attribute> <value>
```

#### Returns

The attribute value if it was set.

#### Arguments

*output\_file* Location of the output file.

*attribute* Specifies the name of the attribute Refer to *File Attribute Properties*.

*value* Value to set the attribute to.

#### Example

```
set_fileset_file_attribute my_file_pkg.sv COMMON_SYSTEMVERILOG_PACKAGE  
my_file_package
```



### 13.1.6.6 get\_fileset\_properties

#### Description

Returns a list of properties that can be set on a fileset.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_fileset_properties
```

#### Returns

A list of property names. Refer to *Fileset Properties*.

#### Arguments

No arguments.

#### Example

```
get_fileset_properties
```

#### Related Links

- [add\\_fileset](#) on page 861
- [get\\_fileset\\_properties](#) on page 866
- [set\\_fileset\\_property](#) on page 863
- [Fileset Properties](#) on page 899



### 13.1.6.7 get\_fileset\_property

#### Description

Returns the value of a fileset property for a fileset.

#### Availability

Main Program, Elaboration, Fileset Generation

#### Usage

```
get_fileset_property <fileset> <property>
```

#### Returns

The value of the property.

#### Arguments

*fileset* The name of the fileset.

*property* The name of the property to query. Refer to *Fileset Properties*.

#### Example

```
get_fileset_property fileset property
```

#### Related Links

[Fileset Properties](#) on page 899



### 13.1.6.8 get\_fileset\_sim\_properties

#### Description

Returns simulator properties for a fileset.

#### Availability

Main Program, Fileset Generation

#### Usage

```
get_fileset_sim_properties <fileset> <platform> <property>
```

#### Returns

The fileset simulator properties.

#### Arguments

*fileset* The name of the fileset.

*platform* The operating system that applies to the property. Refer to *Operating System Properties*.

*property* Specifies the name of the property to set. Refer to *Simulator Properties*.

#### Example

```
get_fileset_sim_properties my_fileset LINUX64 OPT_CADENCE_64BIT
```

#### Related Links

- [add\\_fileset](#) on page 861
- [set\\_fileset\\_sim\\_properties](#) on page 869
- [Operating System Properties](#) on page 911
- [Simulator Properties](#) on page 905



### 13.1.6.9 set\_fileset\_sim\_properties

#### Description

Sets simulator properties for a given fileset

#### Availability

Main Program, Fileset Generation

#### Usage

```
set_fileset_sim_properties <fileset> <platform> <property> <value>
```

#### Returns

The fileset simulator properties if they were set.

#### Arguments

*fileset* The name of the fileset.

*platform* The operating system that applies to the property. Refer to *Operating System Properties*.

*property* Specifies the name of the property to set. Refer to *Simulator Properties*.

*value* Specifies the value of the property.

#### Example

```
set_fileset_sim_properties my_fileset LINUX64 OPT_MENTOR_PLI "{libA} {libB}"
```

#### Related Links

- [get\\_fileset\\_sim\\_properties](#) on page 868
- [Operating System Properties](#) on page 911
- [Simulator Properties](#) on page 905



### 13.1.6.10 create\_temp\_file

#### Description

Creates a temporary file, which you can use inside the fileset callbacks of a `_hw.tcl` file. This temporary file is included in the generation output if it is added using the `add_fileset_file` command.

#### Availability

Fileset Generation

#### Usage

```
create_temp_file <path>
```

#### Returns

The path to the temporary file.

#### Arguments

*path* The name of the temporary file.

#### Example

```
set filelocation [create_temp_file "./hdl/compute_frequency.v" ]
add_fileset_file compute_frequency.v VERILOG PATH ${filelocation}
```

#### Related Links

- [add\\_fileset](#) on page 861
- [add\\_fileset\\_file](#) on page 862



### **13.1.7 Miscellaneous**

- [check\\_device\\_family\\_equivalence](#) on page 872
- [get\\_device\\_family\\_displayname](#) on page 873
- [get\\_qip\\_strings](#) on page 874
- [set\\_qip\\_strings](#) on page 875
- [set\\_interconnect\\_requirement](#) on page 876



### 13.1.7.1 check\_device\_family\_equivalence

#### Description

Returns 1 if the device family is equivalent to one of the families in the device families list., Returns 0 if the device family is not equivalent to any families. This command ignores differences in capitalization and spaces.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
check_device_family_equivalence <device_family> <device_family_list>
```

#### Returns

1 if equivalent, 0 if not equivalent.

#### Arguments

*device\_family* The device family name that is being checked.

*device\_family\_list* The list of device family names to check against.

#### Example

```
check_device_family_equivalence "CYLCONE III LS" { "stratixv" "Cyclone IV"
"cycloneiiiis" }
```

#### Related Links

[get\\_device\\_family\\_displayname](#) on page 873





### 13.1.7.2 get\_device\_family\_displayname

#### Description

Returns the display name of a given device family.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Fileset Generation, Parameter Upgrade

#### Usage

```
get_device_family_displayname <device_family>
```

#### Returns

The preferred display name for the device family.

#### Arguments

*device\_family* A device family name.

#### Example

```
get_device_family_displayname cycloneiils ( returns: "Cyclone IV LS" )
```

#### Related Links

[check\\_device\\_family\\_equivalence](#) on page 872



### 13.1.7.3 get\_qip\_strings

#### Description

Returns a Tcl list of QIP strings for the IP component.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Parameter Upgrade

#### Usage

```
get_qip_strings
```

#### Returns

A Tcl list of qip strings set by this IP component.

#### Arguments

No arguments.

#### Example

```
set strings [ get_qip_strings ]
```

#### Related Links

[set\\_qip\\_strings](#) on page 875



### 13.1.7.4 set\_qip\_strings

#### Description

Places strings in the Intel Quartus Prime IP File (**.qip**) file, which Platform Designer passes to the command as a Tcl list. You add the **.qip** file to your Intel Quartus Prime project on the **Files** page, in the **Settings** dialog box. Successive calls to `set_qip_strings` are not additive and replace the previously declared value.

#### Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Parameter Upgrade

#### Usage

```
set_qip_strings <qip_strings>
```

#### Returns

The Tcl list which was set.

#### Arguments

*qip\_strings* A space-delimited Tcl list.

#### Example

```
set_qip_strings {"QIP Entry 1" "QIP Entry 2"}
```

#### Notes

You can use the following macros in your QIP strings entry:

<code>%entityName%</code>	The generated name of the entity replaces this macro when the string is written to the <b>.qip</b> file.
<code>%libraryName%</code>	The compilation library this IP component was compiled into is inserted in place of this macro inside the <b>.qip</b> file.
<code>%instanceName%</code>	The name of the instance is inserted in place of this macro inside the <b>.qip</b> file.

#### Related Links

[get\\_qip\\_strings](#) on page 874



### 13.1.7.5 set\_interconnect\_requirement

#### Description

Sets the value of an interconnect requirement for a system or an interface on a child instance.

#### Availability

Composition

#### Usage

```
set_interconnect_requirement <element_id> <name> <value>
```

#### Returns

No return value

#### Arguments

*element\_id* {\$system} for system requirements, or qualified name of the interface of an instance, in <instance>.<interface> format. Note that the system identifier has to be escaped in TCL.

*name* The name of the requirement.

*value* The new requirement value.

#### Example

```
set_interconnect_requirement {$system} qsys_mm.maxAdditionalLatency 2
```



## **13.1.8 SystemVerilog Interface Commands**

[add\\_sv\\_interface](#) on page 878

[get\\_sv\\_interfaces](#) on page 879

[get\\_sv\\_interface\\_property](#) on page 880

[get\\_sv\\_interface\\_properties](#) on page 881

[set\\_sv\\_interface\\_property](#) on page 882



### 13.1.8.1 add\_sv\_interface

#### Description

Adds a SystemVerilog interface to the IP component.

#### Availability

Elaboration, Global

#### Usage

```
add_sv_interface <sv_interface_name> <sv_interface_type>
```

#### Returns

No return value.

#### Arguments

*sv\_interface\_name* The name of the SystemVerilog interface in the IP component.

*sv\_interface\_type* The type of the SystemVerilog interface used by the IP component.

#### Example

```
add_sv_interface my_sv_interface my_sv_interface_type
```



### 13.1.8.2 get\_sv\_interfaces

#### Description

Returns the list of SystemVerilog interfaces in the IP component.

#### Availability

Elaboration, Global

#### Usage

```
get_sv_interfaces
```

#### Returns

*String[]* Returns the list of SystemVerilog interfaces defined in the IP component.

#### Arguments

No arguments.

#### Example

```
get_sv_interfaces
```



### 13.1.8.3 get\_sv\_interface\_property

#### Description

Returns the value of a single SystemVerilog interface property from the specified interface.

#### Availability

Elaboration, Global

#### Usage

```
get_sv_interface_property <sv_interface_name> <sv_interface_property>
```

#### Returns

*various* The property value.

#### Arguments

*sv\_interface\_name* The name of a SystemVerilog interface of the system.

*sv\_interface\_property* The name of the property. Refer to *System Verilog Interface Properties*.

#### Example

```
get_sv_interface_property my_sv_interface USE_ALL_PORTS
```





### 13.1.8.4 get\_sv\_interface\_properties

#### Description

Returns the names of all the available SystemVerilog interface properties common to all interface types.

#### Availability

Elaboration, Global

#### Usage

```
get_sv_interface_properties
```

#### Returns

*String[]* The list of SystemVerilog interface properties.

#### Arguments

No arguments.

#### Example

```
get_sv_interface_properties
```



### 13.1.8.5 set\_sv\_interface\_property

#### Description

Sets the value of a property on a SystemVerilog interface.

#### Availability

Elaboration, Global

#### Usage

```
set_sv_interface_property <sv_interface_name> <sv_interface_property>  
<value>
```

#### Returns

No return value.

#### Arguments

*interface* The name of a SystemVerilog interface.

*sv\_interface\_property* The name of the property. Refer to *SystemVerilog Interface Properties*.

*value* The property value.

#### Example

```
set_sv_interface_property my_sv_interface USE_ALL_PORTS True
```



## 13.2 Platform Designer \_hw.tcl Property Reference

- [Script Language Properties](#) on page 884
- [Interface Properties](#) on page 885
- [SystemVerilog Interface Properties](#) on page 885
- [Instance Properties](#) on page 886
- [Parameter Properties](#) on page 887
- [Parameter Type Properties](#) on page 889
- [Parameter Status Properties](#) on page 890
- [Port Properties](#) on page 891
- [Direction Properties](#) on page 893
- [Display Item Properties](#) on page 894
- [Display Item Kind Properties](#) on page 895
- [Display Hint Properties](#) on page 896
- [Module Properties](#) on page 897
- [Fileset Properties](#) on page 899
- [Fileset Kind Properties](#) on page 900
- [Callback Properties](#) on page 901
- [File Attribute Properties](#) on page 902
- [File Kind Properties](#) on page 903
- [File Source Properties](#) on page 904
- [Simulator Properties](#) on page 905
- [Port VHDL Type Properties](#) on page 906
- [System Info Type Properties](#) on page 907
- [Design Environment Type Properties](#) on page 909
- [Units Properties](#) on page 910
- [Operating System Properties](#) on page 911
- [Quartus.ini Type Properties](#) on page 912



### 13.2.1 Script Language Properties

Name	Description
TCL	Implements the script in Tcl.

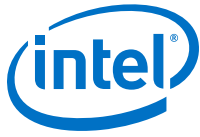


## 13.2.2 Interface Properties

Name	Description
CMSIS_SVD_FILE	Specifies the connection point's associated CMSIS file.
CMSIS_SVD_VARIABLES	Defines the variables inside a .svd file.
ENABLED	Specifies whether or not interface is enabled.
EXPORT_OF	For composed _hwl.tcl files, the EXPORT_OF property indicates which interface of a child instance is to be exported through this interface. Before using this command, you must have created the border interface using add_interface. The interface to be exported is of the form <instanceName.interfaceName>. Example: set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port
PORT_NAME_MAP	A map of external port names to internal port names, formatted as a Tcl list. Example: set_interface_property <interface name> PORT_NAME_MAP "<new port name> <old port name> <new port name 2> <old port name 2>"
SVD_ADDRESS_GROUP	Generates a CMSIS SVD file. Masters in the same SVD address group write register data of their connected slaves into the same SVD file
SVD_ADDRESS_OFFSET	Generates a CMSIS SVD file. Slaves connected to this master have their base address offset by this amount in the SVD file.
SV_INTERFACE	When SV_INTERFACE is set, all the ports in the given interface are part of the SystemVerilog interface. Example: <pre>set_interface_property my_qsys_interface SV_INTERFACE my_sv_interface</pre>

## 13.2.3 SystemVerilog Interface Properties

Name	Description
SV_INTERFACE_TYPE	Set the interface type of the SystemVerilog interface.
USE_ALL_PORTS	When USE_ALL_PORTS is set to true, all the ports defined in the Module, are declared in this SystemVerilog interface. USE_ALL_PORTS must be set to true only if the module has one SystemVerilog interface and the SystemVerilog interface signal names match with the port names declared for Platform Designer interface. When USE_ALL_PORTS is true, SV_INTERFACE_PORT or SV_INTERFACE_SIGNAL port properties should not be set.



### 13.2.4 Instance Properties

Name	Description
HDLINSTANCE_GET_GENERATED_NAME	Platform Designer uses this property to get the auto-generated fixed name when the instance property HDLINSTANCE_USE_GENERATED_NAME is set to true, and only applies to fileSet callbacks.
HDLINSTANCE_USE_GENERATED_NAME	If true, instances added with the <code>add_hdl_instance</code> command are instructed to use unique auto-generated fixed names based on the parameterization.
SUPPRESS_ALL_INFO_MESSAGES	If true, allows you to suppress all Info messages that originate from a child instance.
SUPPRESS_ALL_WARNINGS	If true, allows you to suppress all warnings that originate from a child instance



## 13.2.5 Parameter Properties

Type	Name	Description
Boolean	AFFECTS_ELABORATION	Set AFFECTS_ELABORATION to <code>false</code> for parameters that do not affect the external interface of the module. An example of a parameter that does not affect the external interface is <code>isNonVolatileStorage</code> . An example of a parameter that does affect the external interface is <code>width</code> . When the value of a parameter changes, if that parameter has set <code>AFFECTS_ELABORATION=false</code> , the elaboration phase (calling the callback or hardware analysis) is not repeated, improving performance. Because the default value of <code>AFFECTS_ELABORATION</code> is <code>true</code> , the provided HDL file is normally re-analyzed to determine the new port widths and configuration every time a parameter changes.
Boolean	AFFECTS_GENERATION	The default value of <code>AFFECTS_GENERATION</code> is <code>false</code> if you provide a top-level HDL module; it is <code>true</code> if you provide a fileset callback. Set <code>AFFECTS_GENERATION</code> to <code>false</code> if the value of a parameter does not change the results of fileset generation.
Boolean	AFFECTS_VALIDATION	The <code>AFFECTS_VALIDATION</code> property marks whether a parameter's value is used to set derived parameters, and whether the value affects validation messages. When set to <code>false</code> , this may improve response time in the parameter editor UI when the value is changed.
String[]	ALLOWED_RANGES	Indicates the range or ranges that the parameter value can have. For integers, The <code>ALLOWED_RANGES</code> property is a list of ranges that the parameter can take on, where each range is a single value, or a range of values defined by a start and end value separated by a colon, such as <code>11:15</code> . This property can also specify legal values and display strings for integers, such as <code>{0:None 1:Monophonic 2:Stereo 4:Quadrophonic}</code> meaning 0, 1, 2, and 4 are the legal values. You can also assign display strings to be displayed in the parameter editor for string variables. For example, <code>ALLOWED_RANGES { "dev1:Cyclone IV GX" "dev2:Stratix V GT" }</code> .
String	DEFAULT_VALUE	The default value.
Boolean	DERIVED	When <code>true</code> , indicates that the parameter value can only be set by the IP component, and cannot be set by the user. Derived parameters are not saved as part of an instance's parameter values. The default value is <code>false</code> .
String	DESCRIPTION	A short user-visible description of the parameter, suitable for a tooltip description in the parameter editor.
String[]	DISPLAY_HINT	Provides a hint about how to display a property. The following values are possible: <ul style="list-style-type: none"> <li><code>boolean</code>--for integer parameters whose value can be 0 or 1. The parameter displays as an option that you can turn on or off.</li> <li><code>radio</code>--displays a parameter with a list of values as radio buttons instead of a drop-down list.</li> <li><code>hexadecimal</code>--for integer parameters, display and interpret the value as a hexadecimal number, for example: <code>0x00000010</code> instead of 16.</li> <li><code>fixed_size</code>--for <code>string_list</code> and <code>integer_list</code> parameters, the <code>fixed_size</code> <code>DISPLAY_HINT</code> eliminates the <b>add</b> and <b>remove</b> buttons from tables.</li> </ul>
String	DISPLAY_NAME	This is the GUI label that appears to the left of this parameter.
String	DISPLAY_UNITS	This is the GUI label that appears to the right of the parameter.



Type	Name	Description
Boolean	ENABLED	When <code>false</code> , the parameter is disabled, meaning that it is displayed, but greyed out, indicating that it is not editable on the parameter editor.
String	GROUP	Controls the layout of parameters in GUI
Boolean	HDL_PARAMETER	When true, the parameter must be passed to the HDL IP component description. The default value is <code>false</code> .
String	LONG_DESCRIPTION	A user-visible description of the parameter. Similar to <code>DESCRIPTION</code> , but allows for a more detailed explanation.
String	NEW_INSTANCE_VALUE	This property allows you to change the default value of a parameter without affecting older IP components that have did not explicitly set a parameter value, and use the <code>DEFAULT_VALUE</code> property. The practical result is that older instances continue to use <code>DEFAULT_VALUE</code> for the parameter and new instances use the value that <code>NEW_INSTANCE_VALUE</code> assigns.
String	SV_INTERFACE_PARAMETER	This parameter is used in the SystemVerilog interface instantiation. Example: <pre>set_parameter_property my_parameter SV_INTERFACE_PARAMETER my_sv_interface</pre>
String[]	SYSTEM_INFO	Allows you to assign information about the instantiating system to a parameter that you define. <code>SYSTEM_INFO</code> requires an argument specifying the type of information requested, <code>&lt;info-type&gt;</code> .
String	SYSTEM_INFO_ARG	Defines an argument to be passed to a particular <code>SYSTEM_INFO</code> function, such as the name of a reset interface.
(various)	SYSTEM_INFO_TYPE	Specifies one of the types of system information that can be queried. Refer to <i>System Info Type Properties</i> .
(various)	TYPE	Specifies the type of the parameter. Refer to <i>Parameter Type Properties</i> .
(various)	UNITS	Sets the units of the parameter. Refer to <i>Units Properties</i> .
Boolean	VISIBLE	Indicates whether or not to display the parameter in the parameterization GUI.
String	WIDTH	For a <code>STD_LOGIC_VECTOR</code> parameter, this indicates the width of the logic vector.

**Related Links**

- [System Info Type Properties](#) on page 907
- [Parameter Type Properties](#) on page 889
- [Units Properties](#) on page 910





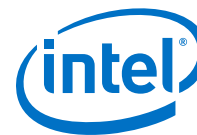
## 13.2.6 Parameter Type Properties

Name	Description
BOOLEAN	A boolean parameter whose value is <code>true</code> or <code>false</code> .
FLOAT	A signed 32-bit floating point parameter. Not supported for HDL parameters.
INTEGER	A signed 32-bit integer parameter.
INTEGER_LIST	A parameter that contains a list of 32-bit integers. Not supported for HDL parameters.
LONG	A signed 64-bit integer parameter. Not supported for HDL parameters.
NATURAL	A 32-bit number that contain values 0 to 2147483647 (0x7fffffff).
POSITIVE	A 32-bit number that contains values 1 to 2147483647 (0x7fffffff).
STD_LOGIC	A single bit parameter whose value can be 1 or 0;
STD_LOGIC_VECTOR	An arbitrary-width number. The parameter property <code>WIDTH</code> determines the size of the logic vector.
STRING	A string parameter.
STRING_LIST	A parameter that contains a list of strings. Not supported for HDL parameters.



### 13.2.7 Parameter Status Properties

Type	Name	Description
Boolean	ACTIVE	Indicates the parameter is a regular parameter.
Boolean	DEPRECATED	Indicates the parameter exists only for backwards compatibility, and may not have any effect.
Boolean	EXPERIMENTAL	Indicates the parameter is experimental, and not exposed in the design flow.



## 13.2.8 Port Properties

Type	Name	Description
(various)	DIRECTION	The direction of the port from the IP component's perspective. Refer to <i>Direction Properties</i> .
String	DRIVEN_BY	Indicates that this output port is always driven to a constant value or by an input port. If all outputs on an IP component specify a <code>driven_by</code> property, the HDL for the IP component will be generated automatically.
String[]	FRAGMENT_LIST	This property can be used in 2 ways: First you can take a single RTL signal and split it into multiple Platform Designer signals <code>add_interface_port &lt;interface&gt; foo &lt;role&gt; &lt;direction&gt; &lt;width&gt; add_interface_port &lt;interface&gt; bar &lt;role&gt; &lt;direction&gt; &lt;width&gt; set_port_property foo fragment_list "my_rtl_signal(3:0)" set_port_property bar fragment_list "my_rtl_signal(6:4)"</code> Second you can take multiple RTL signals and combine them into a single Platform Designer signal <code>add_interface_port &lt;interface&gt; baz &lt;role&gt; &lt;direction&gt; &lt;width&gt; set_port_property baz fragment_list "rtl_signal_1(3:0) rtl_signal_2(3:0)"</code> Note: The listed bits in a port fragment must match the declared width of the Platform Designer signal.
String	ROLE	Specifies an Avalon signal type such as <code>waitrequest</code> , <code>readdata</code> , or <code>read</code> . For a complete list of signal types, refer to the <i>Avalon Interface Specifications</i> .
String	SV_INTERFACE_PORT	This port from the module is used as I/O in the SystemVerilog interface instantiation. The top-level wrapper of the module which contains this port is from the SystemVerilog interface. Example: <pre>set_port_property port_x SV_INTERFACE_PORT my_sv_interface</pre>
String	SV_INTERFACE_PORT_NAME	This property is used only when the Platform Designer port name defined for the module is different from the port name in the SystemVerilog interface. Example: <pre>set_port_property port_x SV_INTERFACE_PORT_NAME port_a</pre> When writing the RTL, the Platform Designer port name <code>port_x</code> is mapped to RTL name <code>port_a</code> in the SystemVerilog interface
String	SV_INTERFACE_SIGNAL	This port from the module is assumed to be inside the SystemVerilog interface or the <code>modport</code> used by the module. The top-level wrapper of the module containing this port is unwrapped from SystemVerilog interface. Example: <pre>set_port_property port_y SV_INTERFACE_SIGNAL my_sv_interface</pre>
String	SV_INTERFACE_SIGNAL_NAME	This property is only used when the Platform Designer port name defined for the module is different from the port name in the SystemVerilog interface. Example: <pre>set_port_property port_y SV_INTERFACE_SIGNAL_NAME port_b</pre>



Type	Name	Description
Boolean	TERMINATION	When <code>true</code> , instead of connecting the port to the Platform Designer system, it is left unconnected for <code>output</code> and <code>bidir</code> or set to a fixed value for <code>input</code> . Has no effect for IP components that implement a generation callback instead of using the default wrapper generation.
BigInteger	TERMINATION_VALUE	The constant value to drive an input port.
(various)	VHDL_TYPE	Indicates the type of a VHDL port. The default value, <code>auto</code> , selects <code>std_logic</code> if the width is fixed at 1, and <code>std_logic_vector</code> otherwise. Refer to <i>Port VHDL Type Properties</i> .
String	WIDTH	The width of the port in bits. Cannot be set directly. Any changes must be set through the <code>WIDTH_EXPR</code> property.
String	WIDTH_EXPR	The width expression of a port. The <code>width_value_expr</code> property can be set directly to a numeric value if desired. When <code>get_port_property</code> is used <code>width</code> always returns the current integer width of the port while <code>width_expr</code> always returns the unevaluated width expression.
Integer	WIDTH_VALUE	The width of the port in bits. Cannot be set directly. Any changes must be set through the <code>WIDTH_EXPR</code> property.

#### Related Links

- [Direction Properties](#) on page 893
- [Port VHDL Type Properties](#) on page 906
- [Avalon Interface Specifications](#)



### 13.2.9 Direction Properties

Name	Description
Bidir	Direction for a bidirectional signal.
Input	Direction for an input signal.
Output	Direction for an output signal.



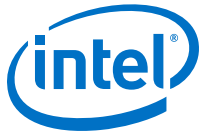
### 13.2.10 Display Item Properties

Type	Name	Description
String	DESCRIPTION	A description of the display item, which you can use as a tooltip.
String[]	DISPLAY_HINT	A hint that affects how the display item displays in the parameter editor.
String	DISPLAY_NAME	The label for the display item in a the parameter editor.
Boolean	ENABLED	Indicates whether the display item is enabled or disabled.
String	PATH	The path to a file. Only applies to display items of type ICON.
String	TEXT	Text associated with a display item. Only applies to display items of type TEXT.
Boolean	VISIBLE	Indicates whether this display item is visible or not.



### 13.2.11 Display Item Kind Properties

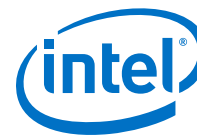
Name	Description
ACTION	An action displays as a button in the GUI. When the button is clicked, it calls the callback procedure. The button label is the display item id.
GROUP	A group that is a child of the <code>parent_group</code> group. If the <code>parent_group</code> is an empty string, this is a top-level group.
ICON	A <code>.gif</code> , <code>.jpg</code> , or <code>.png</code> file.
PARAMETER	A parameter in the instance.
TEXT	A block of text.



### 13.2.12 Display Hint Properties

Name	Description
BIT_WIDTH	Bit width of a number
BOOLEAN	Integer value either 0 or 1.
COLLAPSED	Indicates whether a group is collapsed when initially displayed.
COLUMNS	Number of columns in text field, for example, "columns:N".
EDITABLE	Indicates whether a list of strings allows free-form text entry (editable combo box).
FILE	Indicates that the string is an optional file path, for example, "file:jpg,png,gif".
FIXED_SIZE	Indicates a fixed size for a table or list.
GROW	if set, the widget can grow when the IP component is resized.
HEXADECIMAL	Indicates that the long integer is hexadecimal.
RADIO	Indicates that the range displays as radio buttons.
ROWS	Number of rows in text field, or visible rows in a table, for example, "rows:N".
SLIDER	Range displays as slider.
TAB	if present for a group, the group displays in a tab
TABLE	if present for a group, the group must contain all list-type parameters, which display collectively in a single table.
TEXT	String is a text field with a limited character set, for example, "text:A-Za-z0-9_".
WIDTH	width of a table column





### 13.2.13 Module Properties

Name	Description
ANALYZE_HDL	When set to false, prevents a call to the Intel Quartus Prime mapper to verify port widths and directions, speeding up generation time at the expense of fewer validation checks. If this property is set to false, invalid port widths and directions are discovered during the Intel Quartus Prime software compilation. This does not affect IP components using filesets to manage synthesis files.
AUTHOR	The IP component author.
COMPOSITION_CALLBACK	The name of the composition callback. If you define a composition callback, you cannot not define the generation or elaboration callbacks.
DATASHEET_URL	Deprecated. Use <code>add_documentation_link</code> to provide documentation links.
DESCRIPTION	The description of the IP component, such as "This IP component puts the shizzle in the frobnitz."
DISPLAY_NAME	The name to display when referencing the IP component, such as "My Platform Designer IP Component".
EDITABLE	Indicates whether you can edit the IP component in the Component Editor.
ELABORATION_CALLBACK	The name of the elaboration callback. When set, the IP component's elaboration callback is called to validate and elaborate interfaces for instances of the IP component.
GENERATION_CALLBACK	The name for a custom generation callback.
GROUP	The group in the IP Catalog that includes this IP component.
ICON_PATH	A path to an icon to display in the IP component's parameter editor.
INstantiate_in_System_Module	If true, this IP component is implemented by HDL provided by the IP component. If false, the IP component creates exported interfaces allowing the implementation to be connected in the parent.
INTERNAL	An IP component which is marked as internal does not appear in the IP Catalog. This feature allows you to hide the sub-IP-components of a larger composed IP component.
MODULE_DIRECTORY	The directory in which the hw.tcl file exists.
MODULE_TCL_FILE	The path to the hw.tcl file.
NAME	The name of the IP component, such as <code>my_qsys_component</code> .
OPAQUE_ADDRESS_MAP	For composed IP components created using a <code>_hw.tcl</code> file that include children that are memory-mapped slaves, specifies whether the children's addresses are visible to downstream software tools. When <code>true</code> , the children's address are not visible. When <code>false</code> , the children's addresses are visible.
PREFERRED_SIMULATION_LANGUAGE	The preferred language to use for selecting the fileset for simulation model generation.
REPORT_HIERARCHY	null
STATIC_TOP_LEVEL_MODULE_NAME	Deprecated.



Name	Description
STRUCTURAL_COMPOSITION_CALLBACK	The name of the structural composition callback. This callback is used to represent the structural hierarchical model of the IP component and the RTL can be generated either with module property COMPOSITION_CALLBACK or by ADD_FILESET with target QUARTUS_SYNTH
SUPPORTED_DEVICE_FAMILIES	A list of device family supported by this IP component.
TOP_LEVEL_HDL_FILE	Deprecated.
TOP_LEVEL_HDL_MODULE	Deprecated.
UPGRADEABLE_FROM	null
VALIDATION_CALLBACK	The name of the validation callback procedure.
VERSION	The IP component's version, such as 10.0.



### 13.2.14 Fileset Properties

Name	Description
ENABLE_FILE_OVERWRITE_MODE	null
ENABLE_RELATIVE_INCLUDE_PATHS	If true, HDL files can include other files using relative paths in the fileset.
TOP_LEVEL	The name of the top-level HDL module that the fileset generates. If set, the HDL top level must match the TOP_LEVEL name, and the HDL must not be parameterized. Platform Designer runs the generate callback one time, regardless of the number of instances in the system.



### 13.2.15 Fileset Kind Properties

Name	Description
EXAMPLE_DESIGN	Contains example design files.
QUARTUS_SYNTH	Contains files that Platform Designer uses for the Intel Quartus Prime software synthesis.
SIM_VERILOG	Contains files that Platform Designer uses for Verilog HDL simulation.
SIM_VHDL	Contains files that Platform Designer uses for VHDL simulation.
SYSTEMVERILOG_INTERFACE	This file is treated as SystemVerilog interface file by the Platform Designer. Example: <pre>add_fileset_file mem_ifc.sv SYTEM_VERILOG PATH ".ifc/mem_ifc.sv" SYSTEMVERILOG_INTERFACE</pre>



## 13.2.16 Callback Properties

### Description

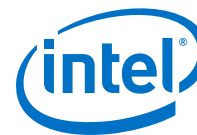
This list describes each type of callback. Each command may only be available in some callback contexts.

Name	Description
ACTION	Called when an ACTION display item's action is performed.
COMPOSITION	Called during instance elaboration when the IP component contains a subsystem.
EDITOR	Called when the IP component is controlling the parameterization editor.
ELABORATION	Called to elaborate interfaces and signals after a parameter change. In API 9.1 and later, validation is called before elaboration. In API 9.0 and earlier, elaboration is called before validation.
GENERATE_VERILOG_SIMULATION	Called when the IP component uses a custom generator to generate the Verilog simulation model for an instance.
GENERATE_VHDL_SIMULATION	Called when the IP component uses a custom generator to generate the VHDL simulation model for an instance.
GENERATION	Called when the IP component uses a custom generator to generate the synthesis HDL for an instance.
PARAMETER_UPGRADE	Called when attempting to instantiate an IP component with a newer version than the saved version. This allows the IP component to upgrade parameters between released versions of the component.
STRUCTURAL_COMPOSITION	Called during instance elaboration when an IP component is represented by a structural hierarchical model which may be different from the generated RTL.
VALIDATION	Called to validate parameter ranges and report problems with the parameter values. In API 9.1 and later, validation is called before elaboration. In API 9.0 and earlier, elaboration is called before validation.



### 13.2.17 File Attribute Properties

Name	Description
ALDEC_SPECIFIC	Applies to Aldec simulation tools and for simulation filesets only.
CADENCE_SPECIFIC	Applies to Cadence simulation tools and for simulation filesets only.
COMMON_SYSTEMVERILOG_PACKAGE	The name of the common SystemVerilog package. Applies to simulation filesets only.
MENTOR_SPECIFIC	Applies to Mentor simulation tools and for simulation filesets only.
SYNOPTIS_SPECIFIC	Applies to Synopsys simulation tools and for simulation filesets only.
TOP_LEVEL_FILE	Contains the top-level module for the fileset and applies to synthesis filesets only.



## 13.2.18 File Kind Properties

Name	Description
DAT	DAT Data
FLI_LIBRARY	FLI Library
HEX	HEX Data
MIF	MIF Data
OTHER	Other
PLI_LIBRARY	PLI Library
SDC	Timing Constraints
SYSTEM_VERILOG	SystemVerilog HDL
SYSTEM_VERILOG_ENCRYPT	Encrypted SystemVerilog HDL
SYSTEM_VERILOG_INCLUDE	SystemVerilog Include
VERILOG	Verilog HDL
VERILOG_ENCRYPT	Encrypted Verilog HDL
VERILOG_INCLUDE	Verilog Include
VHDL	VHDL
VHDL_ENCRYPT	Encrypted VHDL
VPI_LIBRARY	VPI Library



### 13.2.19 File Source Properties

Name	Description
PATH	Specifies the original source file and copies to <code>output_file</code> .
TEXT	Specifies an arbitrary text string for the contents of <code>output_file</code> .





## 13.2.20 Simulator Properties

Name	Description
ENV_ALDEC_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running riviera-pro
ENV_CADENCE_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running ncsim
ENV_MENTOR_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running modelsim
ENV_SYNOPSYS_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running vcs
OPT_ALDEC_PLI	-pli option for riviera-pro
OPT_CADENCE_64BIT	-64bit option for ncsim
OPT_CADENCE_PLI	-loadpli1 option for ncsim
OPT_CADENCE_SVLIB	-sv_lib option for ncsim
OPT_CADENCE_SVROOT	-sv_root option for ncsim
OPT_MENTOR_64	-64 option for modelsim
OPT_MENTOR_CPPPATH	-cpppath option for modelsim
OPT_MENTOR_LDFLAGS	-ldflags option for modelsim
OPT_MENTOR_PLI	-pli option for modelsim
OPT_SYNOPSYS_ACC	+acc option for vcs
OPT_SYNOPSYS_CPP	-cpp option for vcs
OPT_SYNOPSYS_FULL64	-full64 option for vcs
OPT_SYNOPSYS_LDFLAGS	-LDFLAGS option for vcs
OPT_SYNOPSYS_LLIB	-l option for vcs
OPT_SYNOPSYS_VPI	+vpi option for vcs



### 13.2.21 Port VHDL Type Properties

Name	Description
AUTO	The VHDL type of this signal is automatically determined. Single-bit signals are <code>STD_LOGIC</code> ; signals wider than one bit will be <code>STD_LOGIC_VECTOR</code> .
STD_LOGIC	Indicates that the signal is not rendered in VHDL as a <code>STD_LOGIC</code> signal.
STD_LOGIC_VECTOR	Indicates that the signal is rendered in VHDL as a <code>STD_LOGIC_VECTOR</code> signal.



## 13.2.22 System Info Type Properties

Type	Name	Description
String	ADDRESS_MAP	An XML-formatted string describing the address map for the interface specified in the system info argument.
Integer	ADDRESS_WIDTH	The number of address bits required to address all memory-mapped slaves connected to the specified memory-mapped master in this instance, using byte addresses.
String	AVALON_SPEC	The version of the interconnect. SOPC Builder interconnect uses Avalon Specification 1.0. Platform Designer interconnect uses Avalon Specification 2.0.
Integer	CLOCK_DOMAIN	An integer that represents the clock domain for the interface specified in the system info argument. If this instance has interfaces on multiple clock domains, this can be used to determine which interfaces are on each clock domain. The absolute value of the integer is arbitrary.
Long, Integer	CLOCK_RATE	The rate of the clock connected to the clock input specified in the system info argument. If 0, the clock rate is currently unknown.
String	CLOCK_RESET_INFO	The name of this instance's primary clock or reset sink interface. This is used to determine the reset sink to use for global reset when using SOPC interconnect.
String	CUSTOM_INSTRUCTION_SLAVES	Provides custom instruction slave information, including the name, base address, address span, and clock cycle type.
(various)	DESIGN_ENVIRONMENT	A string that identifies the current design environment. Refer to <i>Design Environment Type Properties</i> .
String	DEVICE	The device part number of the currently selected device.
String	DEVICE_FAMILY	The family name of the currently selected device.
String	DEVICE_FEATURES	A list of key/value pairs delineated by spaces indicating whether a particular device feature is available in the currently selected device family. The format of the list is suitable for passing to the Tcl <code>array set</code> command. The keys are device features; the values are 1 if the feature is present, and 0 if the feature is absent.
String	DEVICE_SPEEDGRADE	The speed grade of the currently selected device.
Integer	GENERATION_ID	A integer that stores a hash of the generation time to be used as a unique ID for a generation run.
BigInteger, Long	INTERRUPTS_USED	A mask indicating which bits of an interrupt receiver are connected to interrupt senders. The interrupt receiver is specified in the system info argument.
Integer	MAX_SLAVE_DATA_WIDTH	The data width of the widest slave connected to the specified memory-mapped master.
String, Boolean, Integer	QUARTUS_INI	The value of the quartus.ini setting specified in the system info argument.
Integer	RESET_DOMAIN	An integer that represents the reset domain for the interface specified in the system info argument. If this instance has interfaces on multiple reset domains, this can be used to determine which interfaces are on each reset domain. The absolute value of the integer is arbitrary.



Type	Name	Description
String	TRISTATECONDUIT_INFO	An XML description of the Avalon Tri-state Conduit masters connected to an Avalon Tri-state Conduit slave. The slave is specified as the system info argument. The value contains information about the slave, the connected master instance and interface names, and signal names, directions and widths.
String	TRISTATECONDUIT_MASTERS	The names of the instance's interfaces that are tri-state conduit slaves.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

**Related Links**

[Design Environment Type Properties](#) on page 909

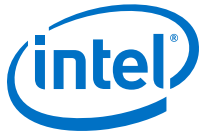


### 13.2.23 Design Environment Type Properties

#### Description

A design environment is used by IP to tell what sort of interfaces are most appropriate to connect in the parent system.

Name	Description
NATIVE	Design environment prefers native IP interfaces.
QSYS	Design environment prefers standard Platform Designer interfaces.



## 13.2.24 Units Properties

Name	Description
Address	A memory-mapped address.
Bits	Memory size, in bits.
BitsPerSecond	Rate, in bits per second.
Bytes	Memory size, in bytes.
Cycles	A latency or count, in clock cycles.
GigabitsPerSecond	Rate, in gigabits per second.
Gigabytes	Memory size, in gigabytes.
Gigahertz	Frequency, in GHz.
Hertz	Frequency, in Hz.
KilobitsPerSecond	Rate, in kilobits per second.
Kilobytes	Memory size, in kilobytes.
Kilohertz	Frequency, in kHz.
MegabitsPerSecond	Rate, in megabits per second.
Megabytes	Memory size, in megabytes.
Megahertz	Frequency, in MHz.
Microseconds	Time, in micros.
Milliseconds	Time, in ms.
Nanoseconds	Time, in ns.
None	Unspecified units.
Percent	A percentage.
Picoseconds	Time, in ps.
Seconds	Time, in s.



### 13.2.25 Operating System Properties

Name	Description
ALL	All operating systems
LINUX32	Linux 32-bit
LINUX64	Linux 64-bit
WINDOWS32	Windows 32-bit
WINDOWS64	Windows 64-bit



### 13.2.26 Quartus.ini Type Properties

Name	Description
ENABLED	Returns 1 if the setting is turned on, otherwise returns 0.
STRING	Returns the string value of the .ini setting.





## 13.3 Document Revision History

The table below indicates edits made to the *Component Interface Tcl Reference* content since its creation.

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>Changed instances of <i>Qsys Pro</i> to Platform Designer</li> <li>Added statement clarifying use of brackets.</li> <li>Added properties and interface commands to support SystemVerilog.</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Implemented Intel rebranding.</li> <li>Implemented Platform Designer rebranding.</li> </ul>
2015.11.02	15.1.0	Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i> .
2015.05.04	15.0.0	Edit to <code>add_fileset_file</code> command.
December 2014	14.1.0	<ul style="list-style-type: none"> <li><code>set_interface_upgrade_map</code></li> <li>Moved <b>Port Roles (Interface Signal Types)</b> section to <i>Platform Designer Interconnect</i>.</li> </ul>
November 2013	13.1.0	<ul style="list-style-type: none"> <li><code>add_hdl_instance</code></li> </ul>
May 2013	13.0.0	<ul style="list-style-type: none"> <li>Consolidated content from other Platform Designer chapters.</li> <li>Added AMBA APB support.</li> </ul>
November 2012	12.1.0	<ul style="list-style-type: none"> <li>Added the <b>demo_axi_memory</b> example with screen shots and example <code>_hw.tcl</code> code.</li> </ul>
June 2012	12.0.0	<ul style="list-style-type: none"> <li>Added AMBA 3 AXI support.</li> <li>Added: <code>set_display_item_property</code>, <code>set_parameter_property</code>, <code>LONG_DESCRIPTION</code>, and static filesets.</li> </ul>
November 2011	11.1.0	<ul style="list-style-type: none"> <li>Template update.</li> <li>Added: <code>set_qip_strings</code>, <code>get_qip_strings</code>, <code>get_device_family_displayname</code>, <code>check_device_family_equivalence</code>.</li> </ul>
May 2011	11.0.0	<ul style="list-style-type: none"> <li>Revised section describing HDL and composed component implementations.</li> <li>Separated reset and clock interfaces in example.</li> <li>Added: <code>TRISTATECONDUIT_INFO</code>, <code>GENERATION_ID</code>, <code>UNIQUE_ID</code>, <code>SYSTEM_INFO</code>.</li> <li>Added: <code>WIDTH</code> and <code>SYSTEM_INFO_ARG</code> parameter properties.</li> <li>Removed the <code>doc_type</code> argument from the <code>add_documentation_link</code> command.</li> <li>Removed: <code>get_instance_parameter_properties</code></li> <li>Added: <code>add_fileset</code>, <code>add_fileset_file</code>, <code>create_temp_file</code>.</li> <li>Updated Tcl examples to show separate clock and reset interfaces.</li> </ul>
December 2010	10.1.0	<ul style="list-style-type: none"> <li>Initial release.</li> </ul>

### Related Links

#### Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 14 Platform Designer System Design Components

---

You can use Platform Designer IP components to create Platform Designer systems. Platform Designer interfaces include components appropriate for streaming high-speed data, reading and writing registers and memory, controlling off-chip devices, and transporting data between components.

*Note:* Intel now refers to Qsys Pro as Platform Designer.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

### Related Links

- [Creating a System with Platform Designer](#) on page 327
- [Platform Designer Interconnect](#) on page 659
- [AMBA Protocol Specifications](#)
- [Embedded Peripherals IP User Guide](#)
- [Avalon Interface Specifications](#)

### 14.1 Bridges

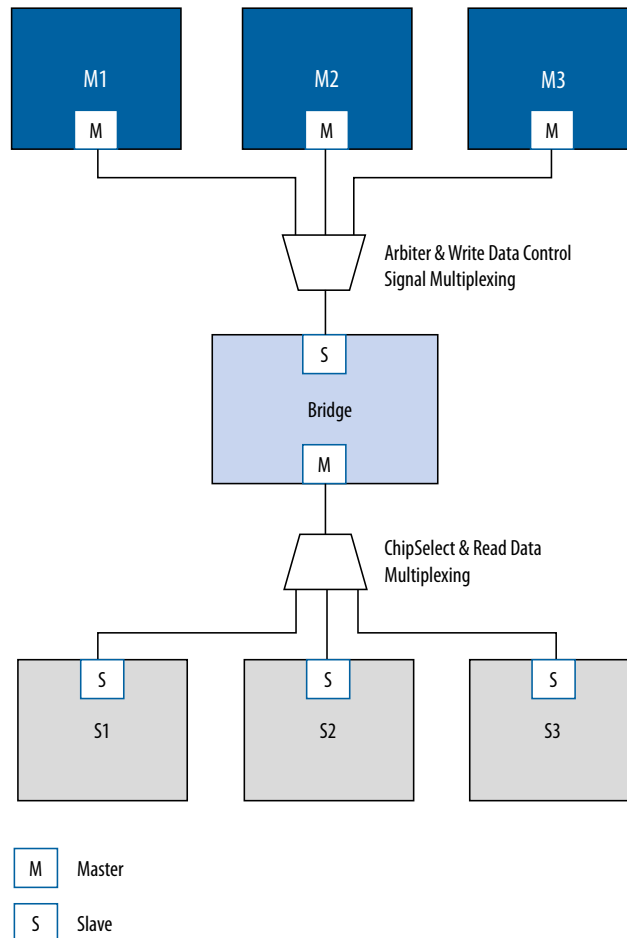
Bridges affect the way Platform Designer transports data between components. You can insert bridges between master and slave interfaces to control the topology of a Platform Designer system, which affects the interconnect that Platform Designer generates. You can also use bridges to separate components into different clock domains to isolate clock domain crossing logic.

A bridge has one slave interface and one master interface. In Platform Designer, one or more master interfaces from other components connect to the bridge slave. The bridge master connects to one or more slave interfaces on other components.



**Figure 279. Using a Bridge in a Platform Designer System**

In this example, three masters have logical connections to three slaves, although physically each master connects only to the bridge. Transfers initiated to the slave propagate to the master in the same order in which the transfers are initiated on the slave.

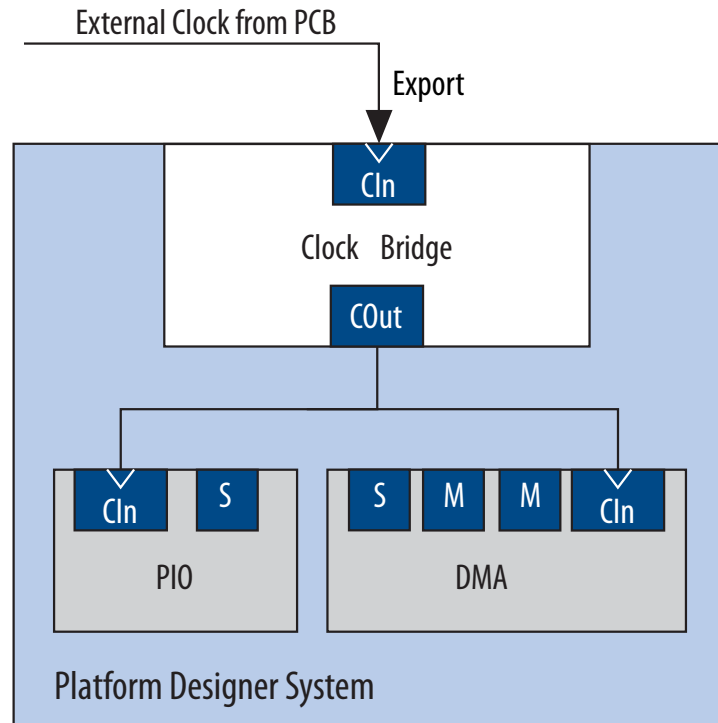


### 14.1.1 Clock Bridge

The Clock Bridge connects a clock source to multiple clock input interfaces. You can use the clock bridge to connect a clock source that is outside the Platform Designer system. Create the connection through an exported interface, and then connect to multiple clock input interfaces.

Clock outputs match fan-out performance without the use of a bridge. You require a bridge only when you want a clock from an exported source to connect internally to more than one source.

Figure 280. Clock Bridge



### 14.1.2 Avalon-MM Clock Crossing Bridge

The Avalon-MM Clock Crossing Bridge transfers Avalon-MM commands and responses between different clock domains. You can also use the Avalon-MM Clock Crossing Bridge between AXI masters and slaves of different clock domains.

The Avalon-MM Clock Crossing Bridge uses asynchronous FIFOs to implement clock crossing logic. The bridge parameters control the depth of the command and response FIFOs in both the master and slave clock domains. If the number of active reads exceeds the depth of the response FIFO, the Clock Crossing Bridge stops sending reads.

To maintain throughput for high-performance applications, increase the response FIFO depth from the default minimum depth, which is twice the maximum burst size.

**Note:** When you use the FIFO-based clock crossing a Platform Designer system, the DC FIFO is automatically inserted in the Platform Designer system. The reset inputs for the DC FIFO connect to the reset sources for the connected master and slave components on either side of the DC FIFO. For this configuration, you must assert both the resets on the master and the slave sides at the same time to ensure the DC FIFO resets properly. Alternatively, you can drive both resets from the same reset source to guarantee that the DC FIFO resets properly.

**Note:** The clock crossing bridge includes appropriate SDC constraints for its internal asynchronous FIFOs. For these SDC constraints to work correctly, do not set false paths on the pointer crossings in the FIFOs. Do not split the bridge's clocks into separate clock groups when you declare SDC constraints; the split has the same effect as setting false paths.

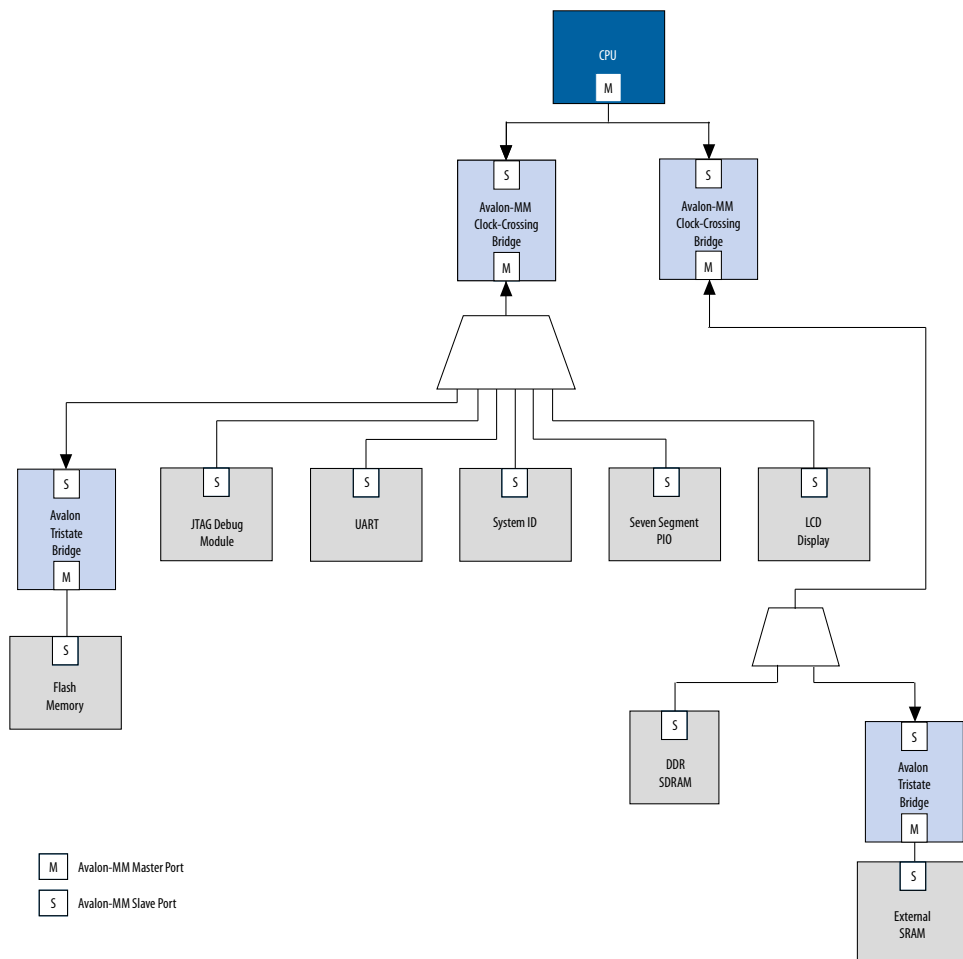


### 14.1.2.1 Avalon-MM Clock Crossing Bridge Example

In the example shown below, the Avalon-MM Clock Crossing bridges separate slave components into two groups. The Avalon-MM Clock Crossing Bridge places the low performance slave components behind a single bridge and clocks the components at a lower speed. The bridge places the high-performance components behind a second bridge and clocks it at a higher speed.

By inserting clock-crossing bridges, you simplify the Platform Designer interconnect and allow the Intel Quartus Prime Fitter to optimize paths that require minimal propagation delay.

Figure 281. Avalon-MM Clock Crossing Bridge



### 14.1.2.2 Avalon-MM Clock Crossing Bridge Parameters

**Table 190. Avalon-MM Clock Crossing Bridge Parameters**

Parameters	Values	Description
<b>Data width</b>	8, 16, 32, 64, 128, 256, 512, 1024 bits	Determines the data width of the interfaces on the bridge, and affects the size of both FIFOs. For the highest bandwidth, set <b>Data width</b> to be as wide as the widest master that connects to the bridge.
<b>Symbol width</b>	1, 2, 4, 8, 16, 32, 64 (bits)	Number of bits per symbol. For example, byte-oriented interfaces have 8-bit symbols.
<b>Address width</b>	1-32 bits	The address bits needed to address the downstream slaves.
<b>Use automatically-determined address width</b>	-	The minimum bridge address width that is required to address the downstream slaves.
<b>Maximum burst size</b>	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 bits	Determines the maximum length of bursts that the bridge supports.
<b>Command FIFO depth</b>	2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 bits	Command (master-to-slave) FIFO depth.
<b>Respond FIFO depth</b>	2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 bits	Slave-to-master FIFO depth.
<b>Master clock domain synchronizer depth</b>	2, 3, 4, 5 bits	The number of pipeline stages in the clock crossing logic in the issuing master to target slave direction. Increasing this value leads to a larger mean time between failures (MTBF). You can determine the MTBF for a design by running a timing analysis.
<b>Slave clock domain synchronizer depth</b>	2, 3, 4, 5 bits	The number of pipeline stages in the clock crossing logic in the target slave to master direction. Increasing this value leads to a larger meantime between failures (MTBF). You can determine the MTBF for a design by running a timing analysis.

### 14.1.3 Avalon-MM Pipeline Bridge

The Avalon-MM Pipeline Bridge inserts a register stage in the Avalon-MM command and response paths. The bridge accepts commands on its slave port and propagates the commands to its master port. The pipeline bridge provides separate parameters to turn on pipelining for command and response signals.

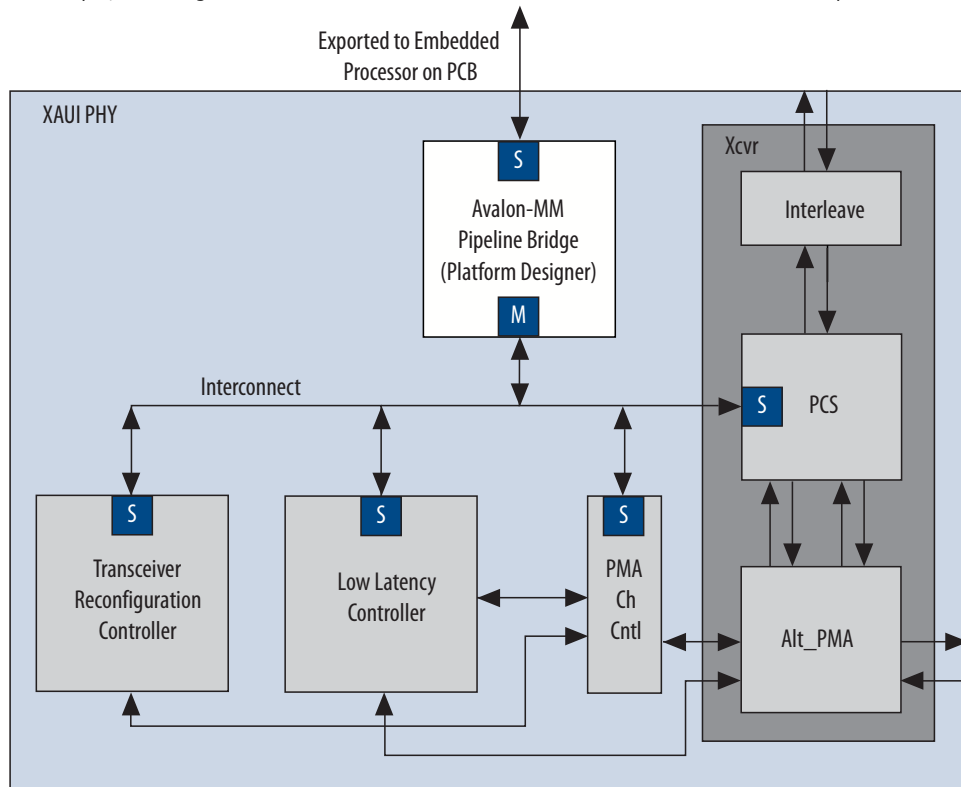
The **Maximum pending read transactions** parameter is the maximum number of pending reads that the Avalon-MM bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected slaves and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon-MM bridge. In general, the value is between 4 and 32. The limit for maximum queued transactions is 64.

You can use the Avalon-MM bridge to export a single Avalon-MM slave interface to control multiple Avalon-MM slave devices. The pipelining feature is optional.



**Figure 282. Avalon-MM Pipeline Bridge in a XAUI PHY Transceiver IP Core**

In this example, the bridge transfers commands received on its slave interface to its master port.

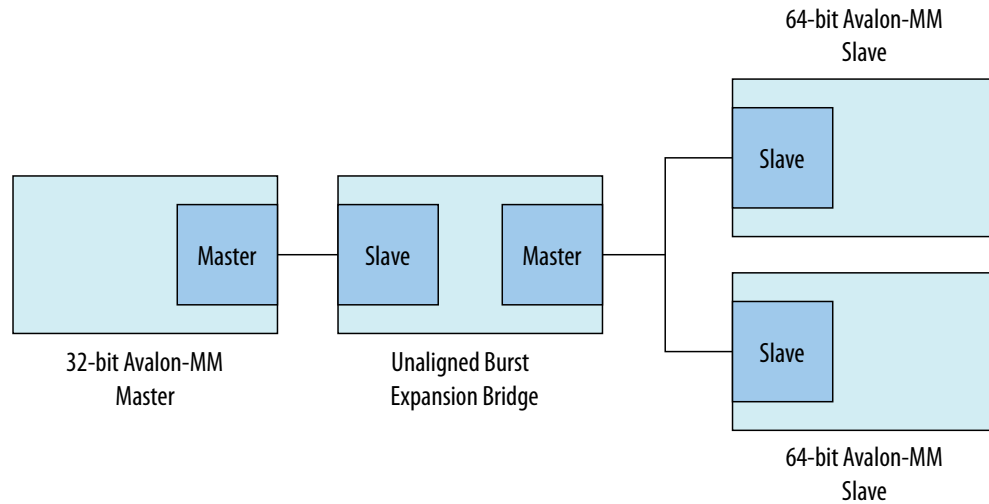


Because the slave interface is exported to the pins of the device, having a single slave port, rather than separate ports for each slave device, reduces the pin count of the FPGA.

#### 14.1.4 Avalon-MM Unaligned Burst Expansion Bridge

The Avalon-MM Unaligned Burst Expansion Bridge aligns read burst transactions from masters connected to its slave interface, to the address space of slaves connected to its master interface. This alignment ensures that all read burst transactions are delivered to the slave as a single transaction.

**Figure 283. Avalon-MM Unaligned Burst Expansion Bridge**



You can use the Avalon Unaligned Burst Expansion Bridge to align read burst transactions from masters that have narrower data widths than the target slaves. Using the bridge for this purpose improves bandwidth utilization for the master-slave pair, and ensures that unaligned bursts are processed as single transactions rather than multiple transactions.

*Note:* Do not use the Avalon-MM Unaligned Burst Expansion Bridge if any connected slave has read side effects from reading addresses that are exposed to any connected master's address map. This bridge can cause read side effects due to alignment modification to read burst transaction addresses.

*Note:* The Avalon-MM Unaligned Burst Expansion Bridge does not support VHDL simulation.

#### 14.1.4.1 Using the Avalon-MM Unaligned Burst Expansion Bridge

When a master sends a read burst transaction to a slave, the Avalon-MM Unaligned Burst Expansion Bridge initially determines whether the start address of the read burst transaction is aligned to the slave's memory address space. If the base address is aligned, the bridge does not change the base address. If the base address is not aligned, the bridge aligns the base address to the nearest aligned address that is less than the requested base address.

The Avalon-MM Unaligned Burst Expansion Bridge then determines whether the final word requested by the master is the last word at the slave read burst address. If a single slave address contains multiple words, all those words must be requested for a single read burst transaction to occur.

- If the final word requested by the master is the last word at the slave read burst address, the bridge does not modify the burst length of the read burst command to the slave.
- If the final word requested by the master is not the last word at the slave read burst address, the bridge increases the burst length of the read burst command to the slave. The final word requested by the modified read burst command is then the last word at the slave read burst address.





The bridge stores information about each aligned read burst command that it sends to slaves connected to a master interface. When a read response is received on the master interface, the bridge determines if the base address or burst length of the issued read burst command was altered.

If the bridge alters either the base address or the burst length of the issued read burst command, it receives response words that the master did not request. The bridge suppresses words that it receives from the aligned burst response that are not part of the original read burst command from the master.

### 14.1.4.2 Avalon-MM Unaligned Burst Expansion Bridge Parameters

Figure 284. Avalon-MM Unaligned Burst Expansion Bridge Parameter Editor

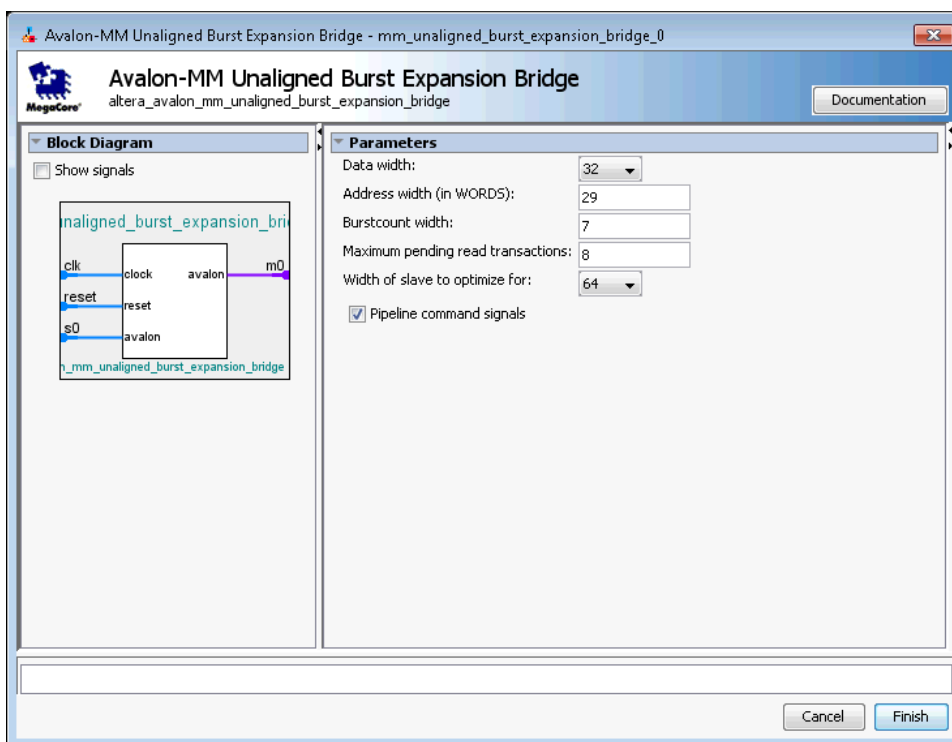


Table 191. Avalon-MM Unaligned Burst Expansion Bridge Parameters

Parameter	Description
<b>Data width</b>	Data width of the master connected to the bridge.
<b>Address width (in WORDS)</b>	The address width of the master connected to the bridge.
<b>Burstcount width</b>	The burstcount signal width of the master connected to the bridge.
<b>Maximum pending read transactions</b>	The <b>Maximum pending read transactions</b> parameter is the maximum number of pending reads that the Avalon-MM bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected slaves and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon-MM bridge. In general, the value is between 4 and 32. The limit for maximum queued transactions is 64.
<b>Width of slave to optimize for</b>	The data width of the connected slave. Supported values are: 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 bits.

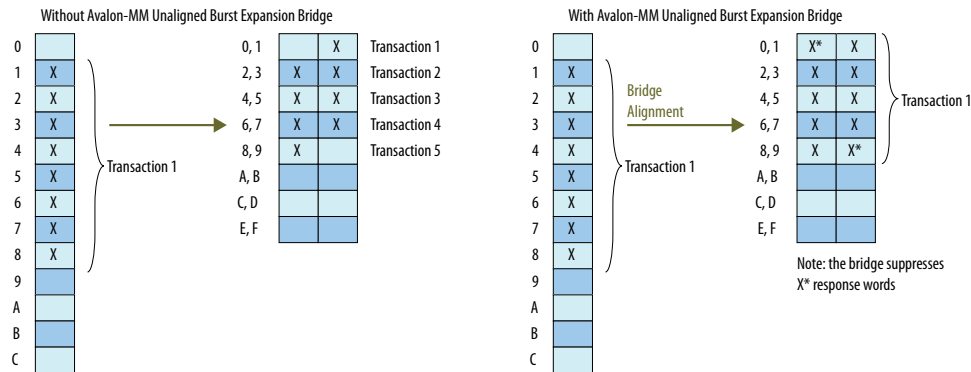
*continued...*

Parameter	Description
	<i>Note:</i> If you connect multiple slaves, all slaves must have the same data width.
<b>Pipeline command signals</b>	When turned on, the command path is pipelined, minimizing the bridge's critical path at the expense of increased logic usage and latency.

### 14.1.4.3 Avalon-MM Unaligned Burst Expansion Bridge Example

**Figure 285. Unaligned Burst Expansion Bridge**

The example below shows an unaligned read burst command from a master that the Avalon-MM Unaligned Burst Expansion Bridge converts to an aligned request for a connected slave, and the suppression of words due to the aligned read burst command. In this example, a 32-bit master requests an 8-beat burst of 32-bit words from a 64-bit slave with a start address that is not 64-bit aligned.



Because the target slave has a 64-bit data width, address 1 is unaligned in the slave's address space. As a result, several smaller burst transactions are needed to request the data associated with the master's read burst command.

With an Avalon-MM Unaligned Burst Expansion Bridge in place, the bridge issues a new read burst command to the target slave beginning at address 0 with burst length 10, which requests data up to the word stored at address 9.

When the bridge receives the word corresponding to address 0, it suppresses it from the master, and then delivers the words corresponding to addresses 1 through 8 to the master. When the bridge receives the word corresponding to address 9, it suppresses that word from the master.

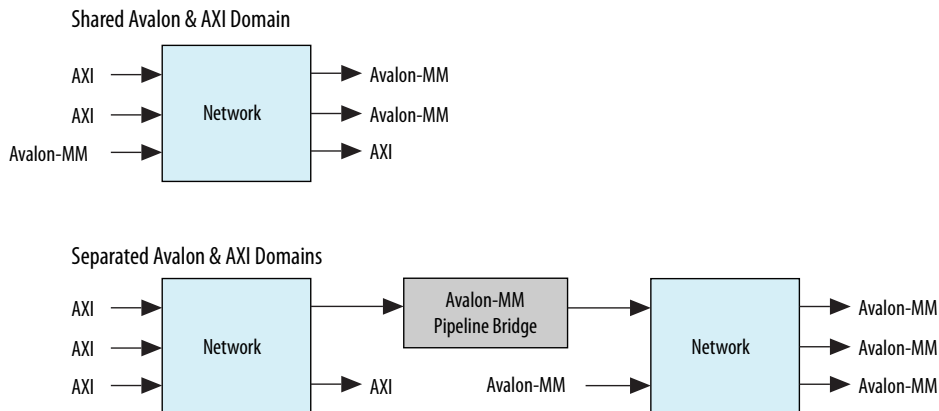
### 14.1.5 Bridges Between Avalon and AXI Interfaces

When designing a Platform Designer system, you can make connections between AXI and Avalon interfaces without the use of explicitly-instantiated bridges; the interconnect provides all necessary bridging logic. However, this does not prevent the use of explicit bridges to separate the AXI and Avalon domains.



**Figure 286. Avalon-MM Pipeline Bridge Between Avalon-MM and AXI Domains**

Using an explicit Avalon-MM bridge to separate the AXI and Avalon domains reduces the amount of bridging logic in the interconnect at the expense of concurrency.



### 14.1.6 AXI Bridge

With an AXI bridge, you can influence the placement of resource-intensive components, such as the width and burst adapters. Depending on its use, an AXI bridge may reduce throughput and concurrency, in return for higher  $f_{MAX}$  and less logic.

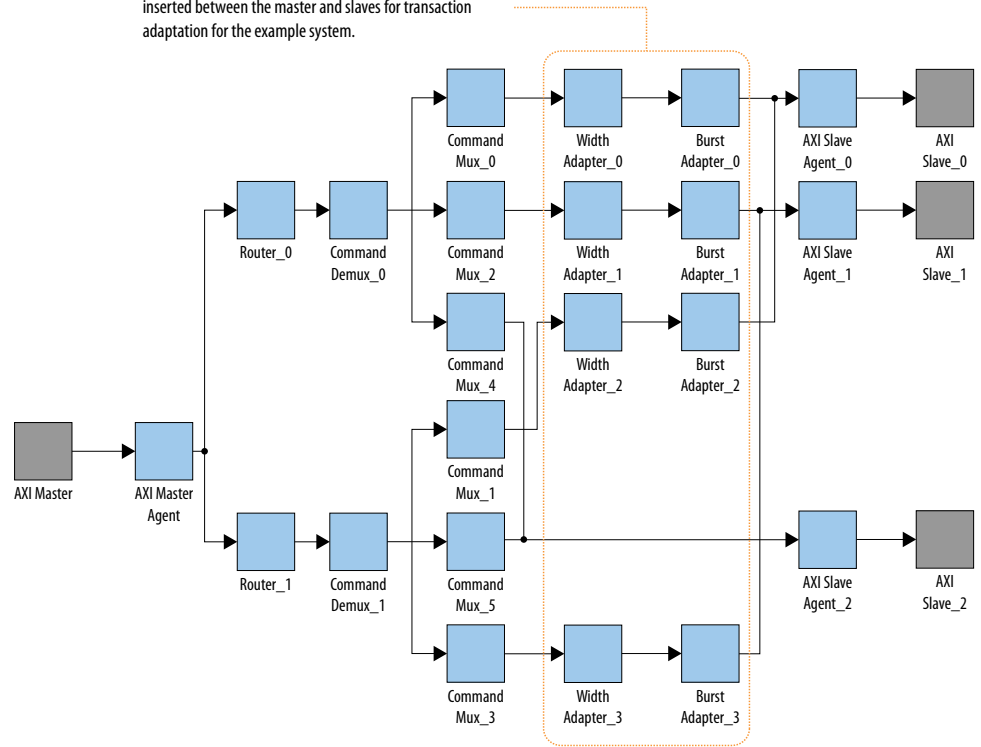
You can use an AXI bridge to group different parts of your Platform Designer system. Other parts of the system can then connect to the bridge interface instead of to multiple separate master or slave interfaces. You can also use an AXI bridge to export AXI interfaces from Platform Designer systems.

**Example 103. Reducing the Number of Adapters by Adding a Bridge**

The figure shows a system with a single AXI master and three AXI slaves. It also has various interconnect components, such as routers, demultiplexers, and multiplexers. Two of the slaves have a narrower data width than the master; 16-bit slaves versus a 32-bit master.

**Figure 287. AXI System Without a Bridge**

Four width adapters (0 - 3) and four burst adapters (0 - 3) are inserted between the master and slaves for transaction adaptation for the example system.

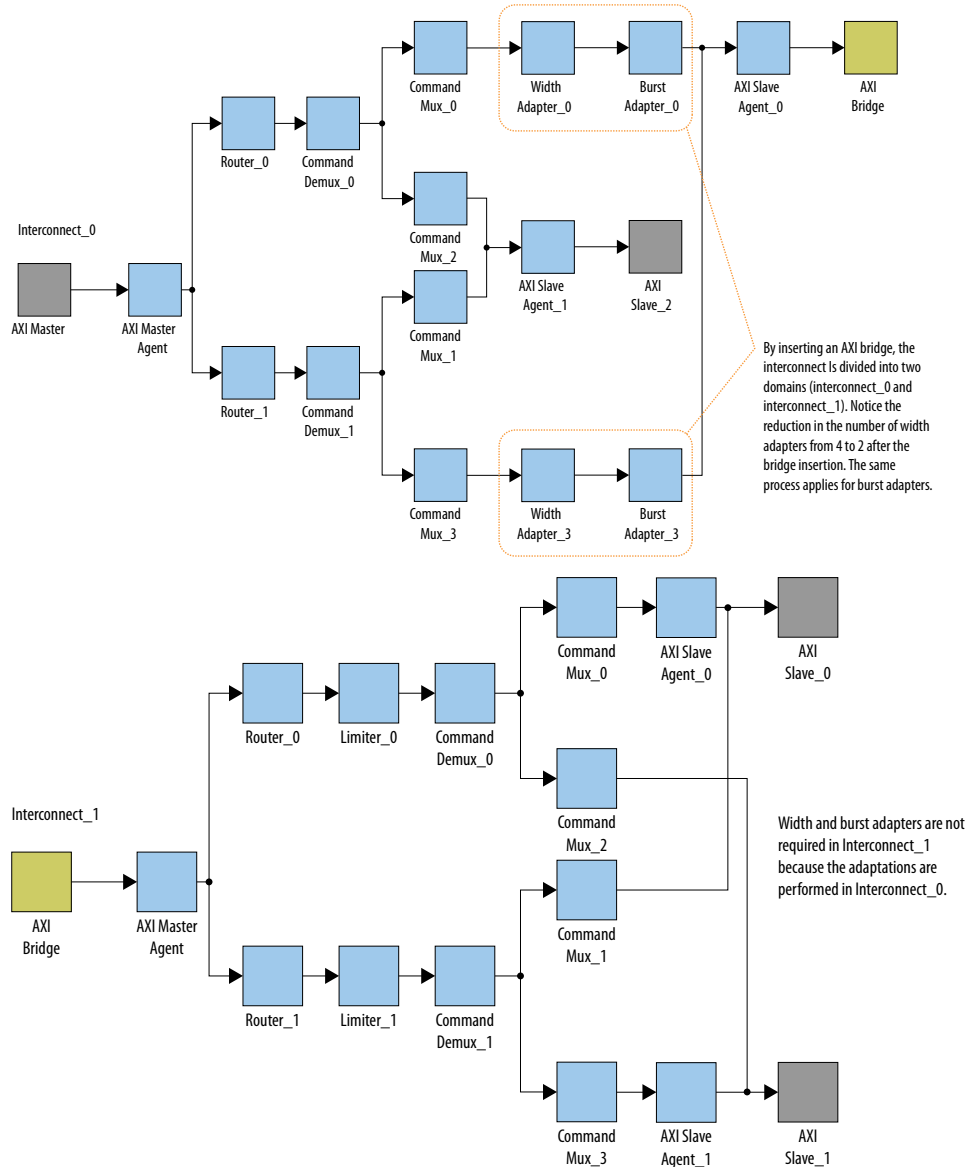


In this system, Platform Designer interconnect creates four width adapters and four burst adapters to access the two slaves.

You can improve resource usage by adding an AXI bridge. Then, Platform Designer needs to add only two width adapters and two burst adapters; one pair for the read channels, and another pair for the write channel.



**Figure 288. Width and Burst Adapters Added to System With a Bridge**



The figure shows the same system with an AXI bridge component, and the decrease in the number of width and burst adapters. Platform Designer creates only two width adapters and two burst adapters, as compared to the four width adapters and four burst adapters in the previous figure.

Even though this system includes more components, the overall system performance improves because there are fewer resource-intensive width and burst adapters.

### 14.1.6.1 AXI Bridge Signal Types

Based on parameter selections that you make for the AXI Bridge component, Platform Designer instantiates either the AMBA 3 AXI or AMBA 3 AXI master and slave interfaces into the component.



**Note:** In AMBA 3 AXI, aw/aruser accommodates sideband signal usage by hard processor systems (HPS).

**Table 192. Sets of Signals for the AXI Bridge Based on the Protocol**

Signal Name	AMBA 3 AXI	AMBA 3 AXI
awid / arid	yes	yes
awaddr / araddr	yes	yes
awlen / arlen	yes (4-bit)	yes (8-bit)
awsiz e / arsize	yes	yes
awburst / arburst	yes	yes
awlock / arlock	yes	yes (1-bit optional)
awcache / arcache	yes (2-bit)	yes (optional)
awprot / arprot	yes	yes
awuser / aruser	yes	yes
awvalid / arvalid	yes	yes
awready / arready	yes	yes
awqos / arqos	no	yes
awregion / arregion	no	yes
wid	yes	no (optional)
wdata / rdata	yes	yes
wstrb	yes	yes
wlast / rvalid	yes	yes
wvalid / rlast	yes	yes
wready / rready	yes	yes
wuser / ruser	no	yes
bid / rid	yes	yes
bresp / rresp	yes	yes (optional)
bvalid	yes	yes
bready	yes	yes

### 14.1.6.2 AXI Bridge Parameters

In the parameter editor, you can customize the parameters for the AXI bridge according to the requirements of your design.



Figure 289. AXI Bridge Parameter Editor

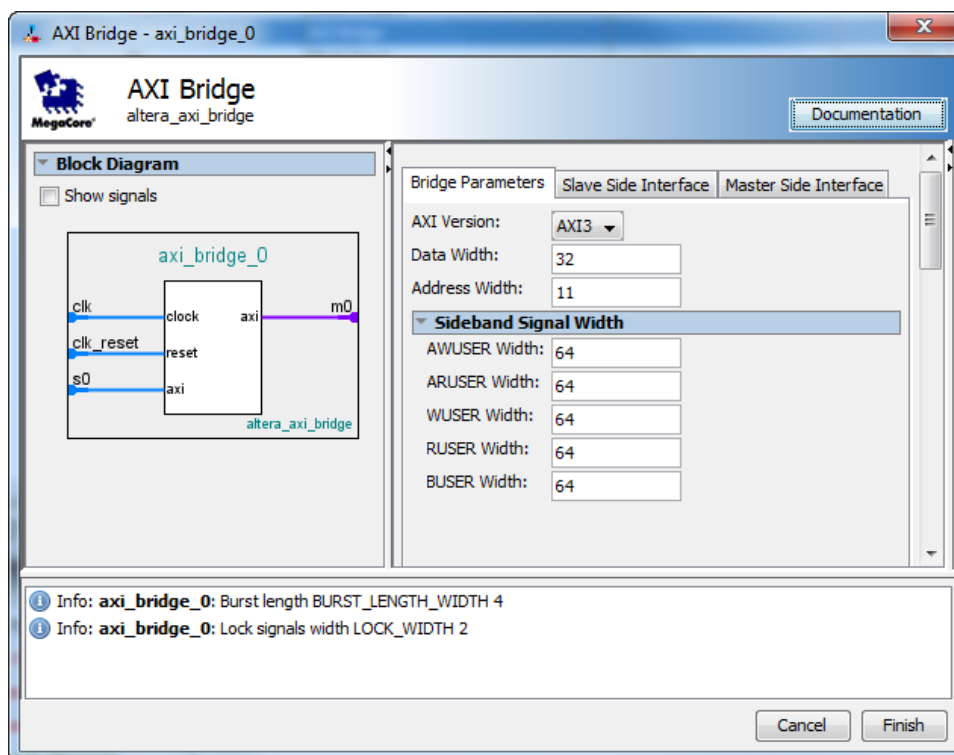


Table 193. AXI Bridge Parameters

Parameter	Type	Range	Description
<b>AXI Version</b>	string	AMBA 3 AXI or AMBA 3 AXI	Specifies the AXI version and signals that Platform Designer generates for the slave and master interfaces of the bridge.
<b>Data Width</b>	integer	8:1024	Controls the width of the data for the master and slave interfaces.
<b>Address Width</b>	integer	1-64 bits	Controls the width of the address for the master and slave interfaces.
<b>AWUSER Width</b>	integer	1-64 bits	Controls the width of the write address channel sideband signals of the master and slave interfaces.
<b>ARUSER Width</b>	integer	1-64 bits	Controls the width of the read address channel sideband signals of the master and slave interfaces.
<b>WUSER Width</b>	integer	1-64 bits	Controls the width of the write data channel sideband signals of the master and slave interfaces.
<b>RUSER Width</b>	integer	1-16 bits	Controls the width of the read data channel sideband signals of the master and slave interfaces.
<b>BUSER Width</b>	integer	1-16 bits	Controls the width of the write response channel sideband signals of the master and slave interfaces.

### 14.1.6.3 AXI Bridge Slave and Master Interface Parameters

**Table 194. AXI Bridge Slave and Master Interface Parameters**

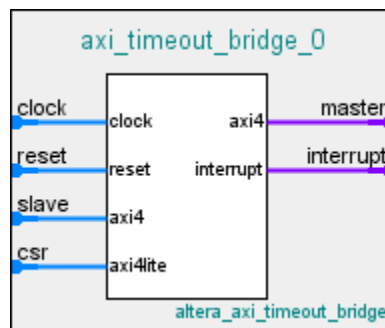
Parameter	Description
<b>ID Width</b>	Controls the width of the thread ID of the master and slave interfaces.
<b>Write/Read/Combined Acceptance Capability</b>	Controls the depth of the FIFO that Platform Designer needs in the interconnect agents based on the maximum pending commands that the slave interface accepts.
<b>Write/Read/Combined Issuing Capability</b>	Controls the depth of the FIFO that Platform Designer needs in the interconnect agents based on the maximum pending commands that the master interface issues. Issuing capability must follow acceptance capability to avoid unnecessary creation of FIFOs in the bridge.

*Note:* Maximum acceptance/issuing capability is a model-only parameter and does not influence the bridge HDL. The bridge does not backpressure when this limit is reached. Downstream components or the interconnect must apply backpressure.

### 14.1.7 AXI Timeout Bridge

The AXI Timeout Bridge allows your system to recover when it freezes, and facilitates debugging. You can place an AXI Timeout Bridge between a single master and a single slave if you know that the slave may time out and cause your system to freeze. If a slave does not accept a command or respond to a command it accepted, its master can wait indefinitely.

**Figure 290. AXI Timeout Bridge**



For a domain with multiple masters and slaves, placement of an AXI Timeout Bridge in your design may be beneficial in the following scenarios:

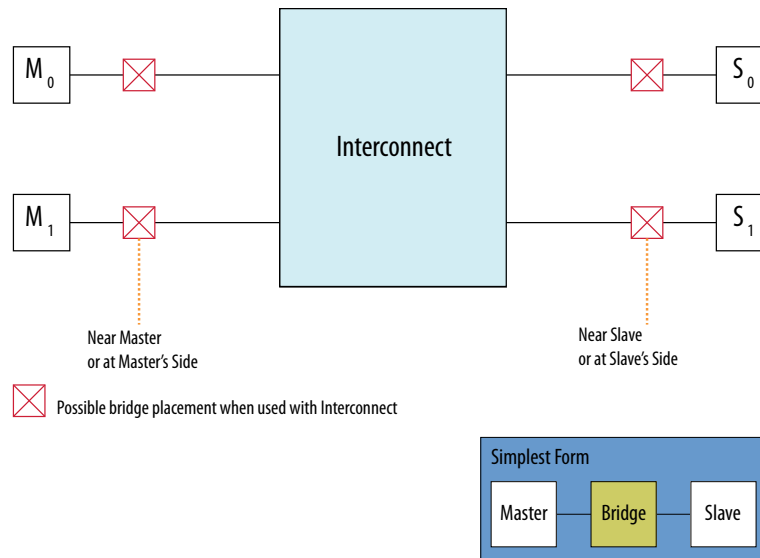
- To recover from a freeze, place the bridge near the slave. If the master attempts to communicate with a slave that freezes, the AXI Timeout Bridge frees the master by generating error responses. The master is then able to communicate with another slave.
- When debugging your system, place the AXI Timeout Bridge near the master. This placement enables you to identify the origin of the burst, and to obtain the full address from the master. Additionally, placing an AXI Timeout Bridge near the master enables you to identify the target slave for the burst.

*Note:* If you place the bridge at the slave's side and you have multiple slaves connected to the same master, you do not get the full address.





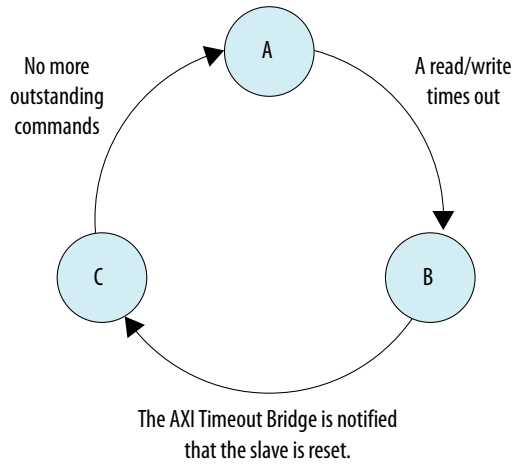
Figure 291. AXI Timeout Bridge Placement



### 14.1.7.1 AXI Timeout Bridge Stages

A timeout occurs when the internal timer in the bridge exceeds the specified number of cycles within which a burst must complete from start to end.

Figure 292. AXI Timeout Bridge Stages



- Ⓐ Slave is functional - The bridge passes through all bursts.
- Ⓑ Slave is unresponsive - The bridge accepts commands and responds (with errors) to commands for the unresponsive slave. Commands are not passed through to the slave at this stage.
- Ⓒ Slave is reset - The bridge does not accept new commands, and responds only to commands that are outstanding.

- When a timeout occurs, the AXI Timeout Bridge asserts an interrupt and reports the burst that caused the timeout to the Configuration and Status Register (CSR).
- The bridge then generates error responses back to the master on behalf of the unresponsive slave. This stage frees the master and certifies the unresponsive slave as dysfunctional.
- The AXI Timeout Bridge accepts subsequent write addresses, write data, and read addresses to the dysfunctional slave. The bridge does not accept outstanding write responses, and read data from the dysfunctional slave is not passed through to the master.
- The `awvalid`, `wvalid`, `bready`, `arvalid`, and `rready` ports are held low at the master interface of the bridge.

**Note:** After a timeout, `awvalid`, `wvalid`, and `arvalid` may be dropped before they are accepted by `awready` at the master interface. While the behavior violates the AXI specification, it occurs only on an interface connected to the slave which has been certified dysfunctional by the AXI Timeout Bridge.

Write channel refers to the AXI write address, data and response channels. Similarly, read channel refers to the AXI read address and data channels. AXI write and read channels are independent of each other. However, when a timeout occurs on either channel, the bridge generates error responses on both channels.



**Table 195. Burst Start and End Definitions for the AXI Timeout Bridge**

Channel	Start	End
Write	When an address is issued. First cycle of <code>awvalid</code> , even if data of the same burst is issued before the address (first cycle of <code>wvalid</code> ).	When the response is issued. First cycle of <code>bvalid</code> .
Read	When an address is issued. First cycle of <code>arvalid</code> .	When the last data is issued. First cycle of <code>rvalid</code> and <code>rlast</code> .

The AXI Timeout Bridge has four required interfaces: Master, Slave, Configuration and Status Register (CSR) (AMBA 3 AXI-Lite), and Interrupt. Platform Designer allows the AXI Timeout Bridge to connect to any AMBA 3 AXI, AMBA 3 AXI, or Avalon master or slave interface. Avalon masters must utilize the bridge’s interrupt output to detect a timeout.

The bridge slave interface accepts write addresses, write data, and read addresses, and then generates the `SLVERR` response at the write response and read data channels. Do not use `buser`, `rdata` and `ruser` at this stage of processing.

To resume normal operation, the dysfunctional slave must be reset and the bridge notified of the change in status via the CSR. Once the CSR notifies the bridge that the slave is ready, the bridge does not accept new commands until all outstanding bursts are responded to with an error response.

The CSR has a 4-bit address width and a 32-bit data width. The CSR reports status and address information when the bridge asserts an interrupt.

**Table 196. CSR Interrupt Status Information for the AXI Timeout Bridge**

Address	Attribute	Name
0x0	write-only	Slave is reset
0x4	read-only	Timed out operation
0x8 through 0xF	read-only	Timed out address

### 14.1.7.2 AXI Timeout Bridge Parameters

**Table 197. AXI Timeout Bridge Parameters**

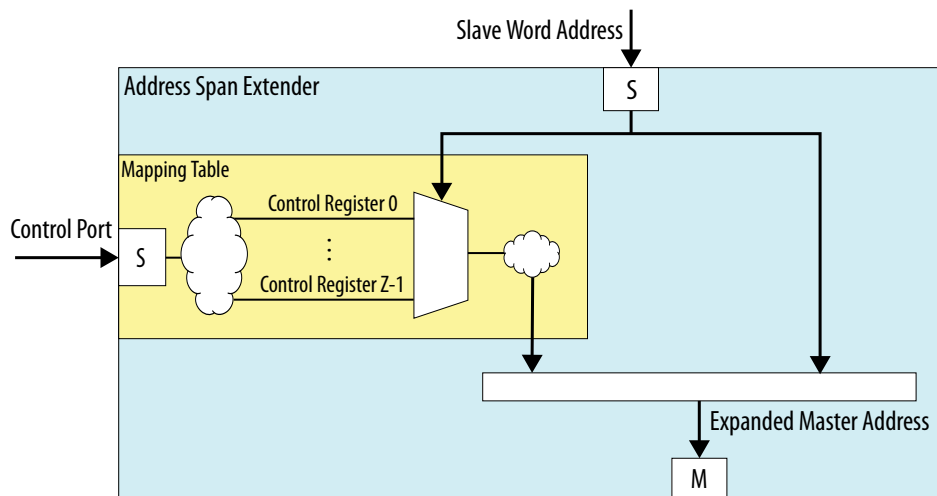
Parameter	Description
<b>ID width</b>	The width of <code>awid</code> , <code>bid</code> , <code>arid</code> , or <code>rid</code> .
<b>Address width</b>	The width of <code>awaddr</code> or <code>araddr</code> .
<b>Data width</b>	The width of <code>wdata</code> or <code>rdata</code> .
<b>User width</b>	The width of <code>awuser</code> , <code>wuser</code> , <code>buser</code> , <code>aruser</code> , or <code>ruser</code> .
<b>Maximum number of outstanding writes</b>	The expected maximum number of outstanding writes.
<b>Maximum number of outstanding reads</b>	The expected maximum number of outstanding reads.
<b>Maximum number of cycles</b>	The number of cycles within which a burst must complete.

### 14.1.8 Address Span Extender

The **Address Span Extender** allows memory-mapped master interfaces to access a larger or smaller address map than the width of their address signals allows. The address span extender splits the addressable space into multiple separate windows, so that the master can access the appropriate part of the memory through the window.

The address span extender does not limit master and slave widths to a 32-bit and 64-bit configuration. You can use the address span extender with 1-64 bit address windows.

**Figure 293. Address Span Extender**



If a processor can address only 2 GB of an address span, and your system contains 4 GB of memory, the address span extender can provide two, 2 GB windows in the 4 GB memory address space. This issue sometimes occurs with Intel SoC devices.

For example, an HPS subsystem in an SoC device can address only 1 GB of an address span within the FPGA, using the HPS-to-FPGA bridge. The address span extender enables the SoC device to address all the address space in the FPGA using multiple 1 GB windows.

**Related Links**

[Platform Designer 64-Bit Addressing Support](#) on page 394

#### 14.1.8.1 CTRL Register Layout

The control registers consist of one 64-bit register for each window, where you specify the window's base address. For example, if `CTRL_BASE` is the base address of the control register, and address span extender contains two windows (0 and 1), then window 0's control register starts at `CTRL_BASE`, and window 1's control register starts at `CTRL_BASE + 8` (using byte addresses).



### 14.1.8.2 Address Span Extender Parameters

**Table 198. Address Span Extender Parameters**

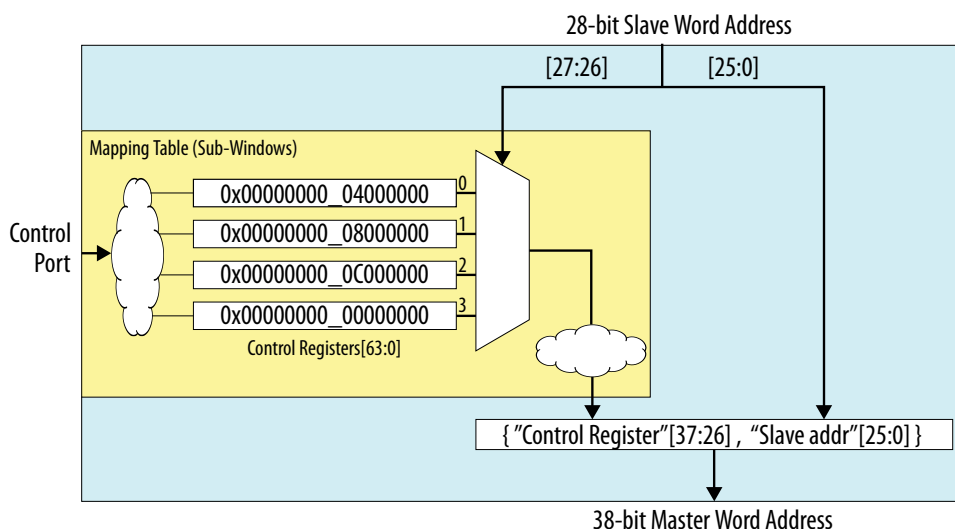
Parameter	Description
<b>Datapath Width</b>	Width of write data and read data signals.
<b>Expanded Master Byte Address Width</b>	Width of the master byte address port. That is, the address span size of all the downstream slaves that attach to the address span extender.
<b>Slave Word Address Width</b>	Width of the slave word address port. That is, the address span size of the downstream slaves that the upstream master accesses.
<b>Burstcount Width</b>	Burst count port width of the downstream slave and the upstream master that attach to the address span extender.
<b>Number of sub-windows</b>	The slave port can represent one or more windows in the master address span. You can subdivide the slave address span into $N$ equal spans in $N$ sub-windows. A remapping register in the CSR slave represents each sub-window, and configures the base address that each sub-window remaps to. The address span extender replaces the upper bits of the address with the stored index value in the remapping register before the master initiates a transaction.
<b>Enable Slave Control Port</b>	Dictates run-time control over the sub-window indexes. If you can define static re-mappings that do not need any change, you do not need to enable this CSR slave.
<b>Maximum Pending Reads</b>	Sets the bridge slave's <code>maximumPendingReadTransactions</code> property. In certain system configurations, you must increase this value to improve performance. This value usually aligns with the properties of the downstream slaves that you attach to this bridge.

### 14.1.8.3 Calculating the Address Span Extender Slave Address

The diagram describes how Platform Designer calculates the slave address. In this example, the address span extender is configured with a 28-bit address space for slaves. The upper 2 bits [27:26] are used to select the control registers.

The lower 26 bits ([25:0]) originate from the address span extender's data port, and are the offset into a particular window.

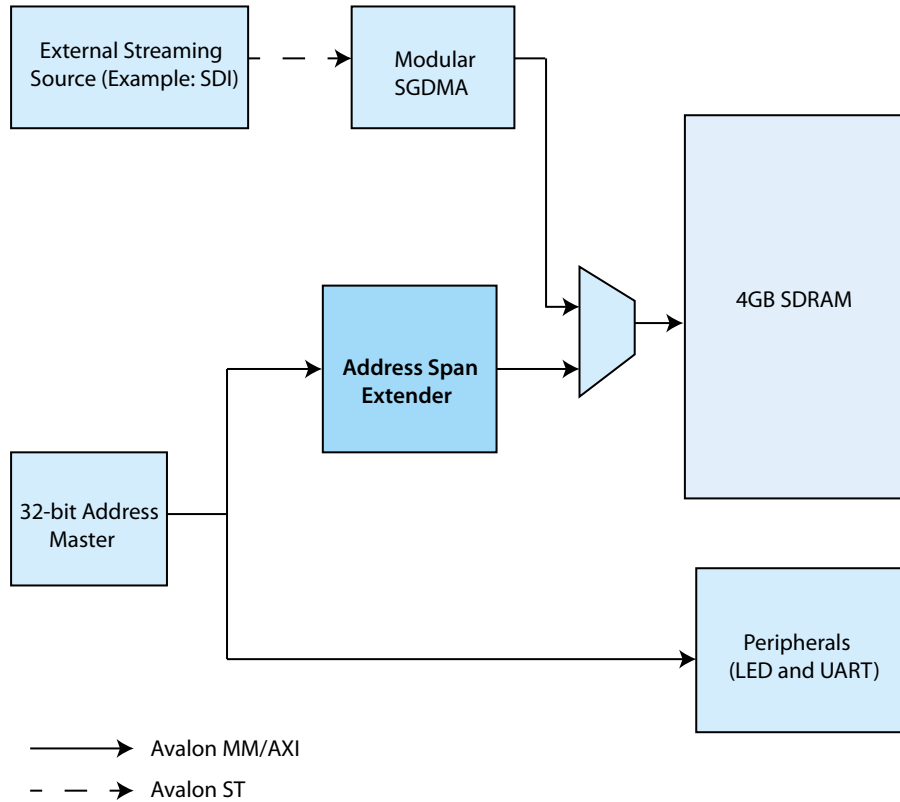
**Figure 294. Address Span Extender**



### 14.1.8.4 Using the Address Span Extender

This example shows when and how to use address span extender component in your Platform Designer design.

**Figure 295. Block Diagram with Address Span Extender**



In the above design, a 32-bit master shares 4 GB SDRAM with an external streaming interface. The master has the path to access streaming data from the SDRAM DDR memory. However, if you connect the whole 32-bit address bus of the master to the SDRAM DDR memory, you cannot connect the master to peripherals such as LED or UART. To avoid this situation, you can implement the address span extender between the master and DDR memory. The address span extender allows the master to access the SDRAM DDR memory and the peripherals at the same time.

To implement address span extender for the above example, you can divide the address window of the address span extender into two sub-windows of 512 MB each. The sub-window 0 is for the master program area. You can dynamically map the sub-window 1 to any area other than the program area.

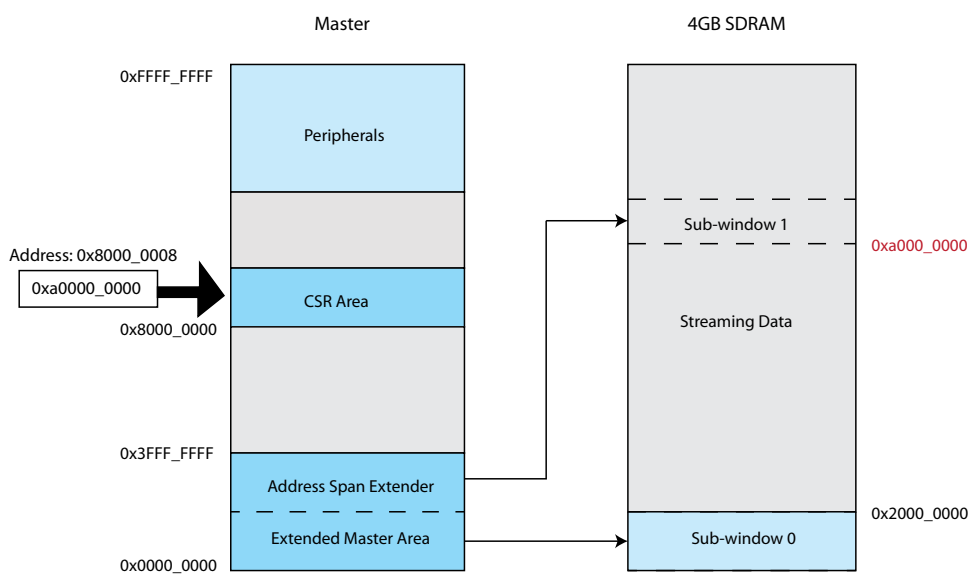
You can change the offset of the address window by setting the base address of sub-window 1 to the control register of the address span extender. However, you must make sure that the sub-window address span masks the base address. You can choose any arbitrary base address. If you set the value 0xa000\_0000 to the control register, Platform Designer maps the sub-window 1 to 0xa000\_0000.



**Table 199. CSR Mapping Table**

Address	Data
0x8000_0000	0x0000_0000
0x8000_0008	0xa000_0000

**Figure 296. Memory mapping for Address Span Extender**

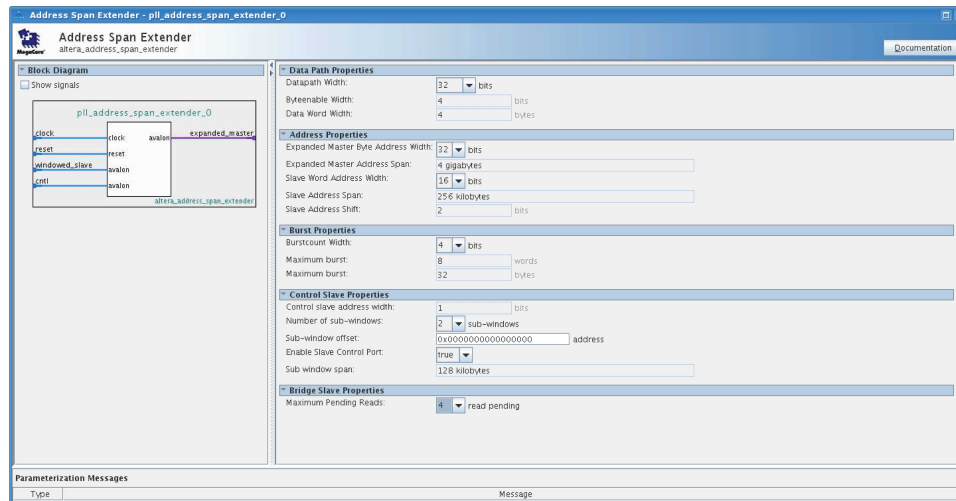


The table below indicates the Platform Designer parameter settings for this address span extender example.

**Table 200. Parameter Settings for the Address Span Extender Example**

Parameter	Value	Description
<b>Datapath Width</b>	32 bits	The CPU has 32-bits data width and the SDRAM DDR memory has 512-bits data width. Since the transaction between the master and SDRAM DDR memory is minimal, set the datapath width to align with the upstream master.
<b>Expanded Master Byte Address</b>	32 bits	The address span extender has a 4 GB address span.
<b>Slave Word Address Width</b>	18 bits	There are two 512 MB sub-windows in reserve for the master. The number of bytes over the data word width in the <b>Datapath Properties</b> (4 bytes for this example) accounts for the slave address.
<b>Burstcount Width</b>	4 bits	The address span extender must handle up to 8 words burst in this example.
<b>Number of sub-windows</b>	2	Address window of the address span extender has two sub-windows of 512 MB each.
<b>Enable Slave Control Port</b>	true	The address span extender component must have control to change the base address of the sub-window.
<b>Maximum Pending Reads</b>	4	This number is the same as SDRAM DDR memory burst count.

Figure 297. Address Span Extender Parameter Editor



**Note:** You can view the address span extender connections in the **System Contents** tab. The windowed slave port and control port connect to the master. The expanded master port connects to the SDRAM DDR memory.

#### 14.1.8.5 Alternate Options for the Address Span Extender

You can set parameters for the address span extender with an initial fixed address value. Enter an address for the **Reset Default for Master Window** option, and select **True** for the **Disable Slave Control Port** option. This allows the address span extender to function as a fixed, non-programmable component.

Each sub-window is equal in size and stacks sequentially in the windowed slave interface's address space. To control the fixed address bits of a particular sub-window, you can write to the sub-window's register in the register control slave interface. Platform Designer structures the logic so that Platform Designer can optimize and remove bits that are not needed.

If **Burstcount Width** is greater than 1, Platform Designer processes the read burst in a single cycle, and assumes all `byteenable` signals are asserted on every cycle.

#### 14.1.8.6 Nios II Support

If the address span extender window is fixed, for example, the **Disable Slave Control Port** option is turned on, then the address span extender performs as a bridge. Components on the slave side of the address span extender that are within the window are visible to the Nios II processor. Components partially within a window appear to the Nios II processor as if they have a reduced span. For example, a memory partially within a window appears as having a smaller size.

You can also use the address span extender to provide a window for the Nios II processor, so that the HPS memory map is visible to the Nios II processor. This technique allows the Nios II processor to communicate with HPS peripherals.

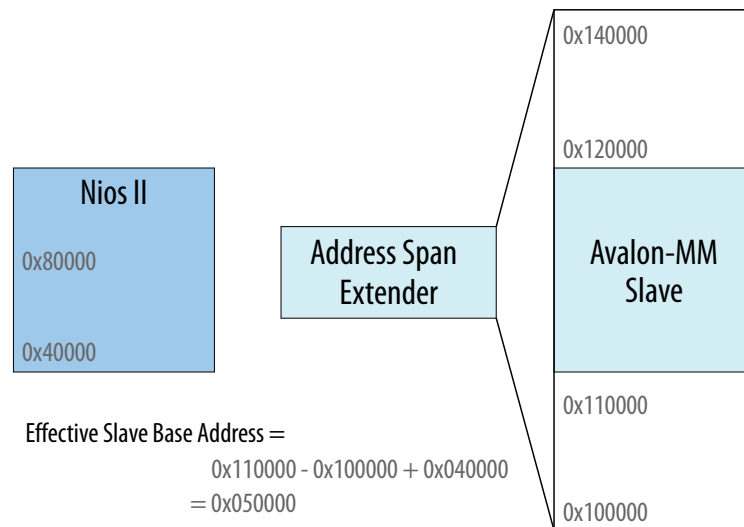




In the example, a Nios II processor has an address span extender from address 0x40000 to 0x80000. There is a window within the address span extender starting at 0x100000. Within the address span extender's address space there is a slave at base address 0x110000. The slave appears to the Nios II processor as being at address:

$$0x110000 - 0x100000 + 0x40000 = 0x050000$$

**Figure 298. Nios II Support and the Address Span Extender**



The address span extender window is dynamic. For example, when the **Disable Slave Control Port** option is turned off, the Nios II processor is unable to see components on the slave side of the address span extender.

## 14.2 Error Response Slave

The Error Response Slave provides a predictable error response service for master interfaces that attempt to access an undefined memory region.

The Error Response Slave is an AMBA 3 AXI component, and appears in the Platform Designer IP Catalog under **Platform Designer Interconnect**.

To comply with the AXI protocol, the interconnect logic must return the `DECERR` error response in cases where the interconnect cannot decode slave access. Therefore, an AXI system with address space not fully decoded to slave interfaces requires the Error Response Slave.

The Error Response Slave behaves like any other component in the system, and connects to other components via translation and adaptation interconnect logic. Connecting an Error Response Slave to masters of different data widths, including Avalon or AXI-Lite masters, can increase resource usage.

An Error Response Slave can connect to clock, reset, and IRQ signals as well as AMBA 3 AXI and AMBA 4 AXI master interfaces without instantiating a bridge. When you connect an Error Response Slave to a master, the Error Response Slave accepts cycles sent from the master, and returns the `DECERR` error response. On the AXI interface, the Error Response Slave supports only a read and write acceptance of capability 1,

and does not support write data interleaving. The Error Response Slave can return responses when simultaneously targeted by a read and write cycle, because its read and write channels are independent.

An optional Avalon interface on the Error Response Slave provides information in a set of CSR registers. CSR registers log the required information when returning an error response.

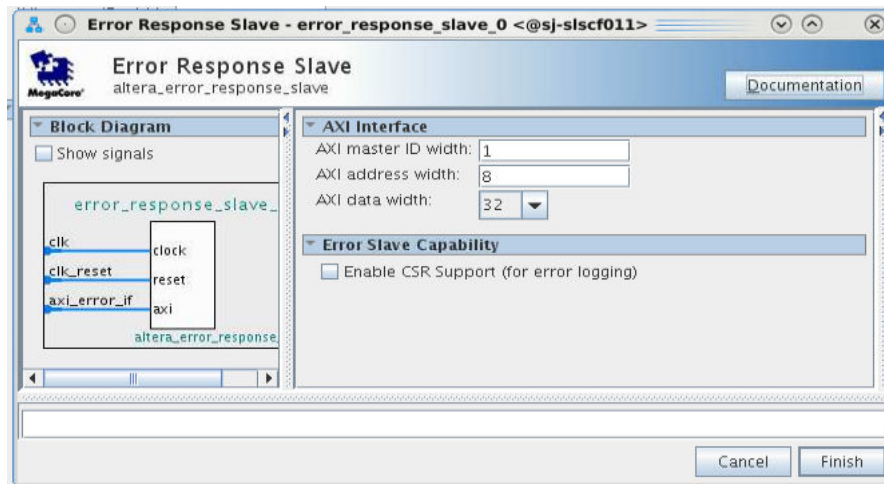
- To set the Error Response Slave as the default slave for a master interface in your system, connect the slave to the master in your Platform Designer system.
- A system can contain more than one Error Response Slave.
- As a best practice, instantiate separate Error Response Slave components for each AXI master in your system.

**Related Links**

- [AMBA 3 AXI Protocol Specification Support \(version 1.0\)](#) on page 718
- [Designating a Default Slave in the System Contents Tab](#) on page 942

**14.2.1 Error Response Slave Parameters**

**Figure 299. Error Response Slave Parameter Editor**



If you turn on **Enable CSR Support (for error logging)** more parameters become available.



Figure 300. Error Response Slave Parameter Editor with Enabled CSR Support

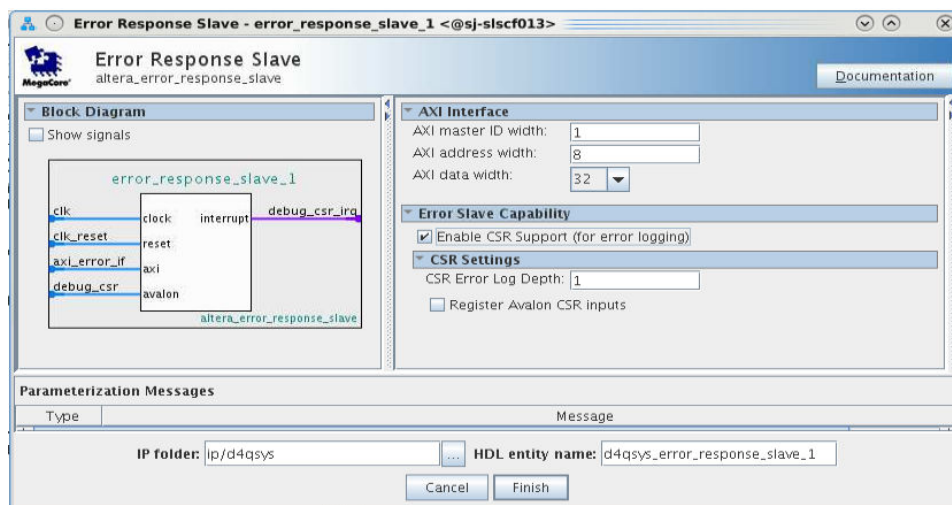


Table 201. Error Response Slave Parameters

Parameter	Value	Description
<b>AXI master ID width</b>	1-8 bits	Specifies the master ID width for error logging.
<b>AXI address width</b>	8-64 bits	Specifies the address width for error logging. This value also affects the overall address width of the system, and should not exceed the maximum address width required in the system.
<b>AXI data width</b>	32, 64, or 128 bits	Specifies the data width for error logging.
<b>Enable CSR Support (for error logging)</b>	On / Off	When turned on, instantiates an Avalon CSR interface for error logging.
<b>CSR Error Log Depth</b>	1-16 bits	Depth of the transaction log, for example, the number of transactions the CSR logs for cycles with errors.
<b>Register Avalon CSR inputs</b>	On / Off	When turned on, controls debug access to the CSR interface.

## 14.2.2 Error Response Slave CSR Registers

The Error Response Slave with enabled CSR support provides a service to handle access violations. This service uses CSR registers for status and logging purposes.

The sequence of actions in the access violation service is equivalent for read and write access violations, but the CSR status bits and log registers are different.

### 14.2.2.1 Error Response Slave Access Violation Service

When an access violation occurs, and the CSR port is enabled:

1. The Error Response Slave generates an interrupt:
  - For a read access violation, the Error Response Slave sets the Read Access Violation Interrupt register bit in the Interrupt Status register.



- For a write access violation, the Error Response Slave sets the `Write Access Violation Interrupt` register bit in the `Interrupt Status` register.
- 2. The Error Response Slave transfers transaction information to the access violation log FIFO. The amount of information that the FIFO can handle is given by the **Error Log Depth** parameter.

You define the **Error Log Depth** in the **Parameter Editor**, when you enable CSR Support.

- 3. Software reads entries of the access violation log FIFO until the corresponding `cycle log valid` bit is cleared, and then exits the service routine.
  - The `Read cycle log valid` bit is in the `Read Access Violation Log CSR Registers`.
  - The `Write cycle log valid` bit is in the `Write Access Violation Log CSR Registers`.
- 4. The Error Response Slave clears the interrupt bit when there are no access violations to report.

Some special cases are:

- If any error occurs when the FIFO is full, the Error Response Slave sets the corresponding `Access Violation Interrupt Overflow` register bit (bits 2 and 3 of the `Status Register` for write and read access violations, respectively). Setting this bit means that not all error entries were written to the access violation log.
- After Software reads an entry in the `Access Violation log`, the Error Response Slave can write a new entry to the log.
- Software can specify the number of entries to read before determining that the access violation service is taking too long to complete, and exit the routine.

### 14.2.2.2 CSR Interrupt Status Registers

**Table 202. CSR Interrupt Status Registers**

For CSR register maps: `Address = Memory Address Base + Offset`.

Offset	Bits	Attribute	Default	Description
0x00	31:4			Reserved.
	3	RW1C	0	Read Access Violation Interrupt Overflow register Asserted when a read access causes the Interconnect to return a DECERR response, and the buffer log depth is full. Indicates that there is a logging error lost due to an exceeded buffer log depth. Cleared by setting the bit to 1.
	2	RW1C	0	Write Access Violation Interrupt Overflow register Asserted when a write access causes the Interconnect to return a DECERR response, and the buffer log depth is full. Indicates that there is a logging error lost due to an exceeded buffer log depth. Cleared by setting the bit to 1.
	1	RW1C	0	Read Access Violation Interrupt register Asserted when a read access causes the Interconnect to return a DECERR response. Cleared by setting the bit to 1.

*continued...*



Offset	Bits	Attribute	Default	Description
				<i>Note:</i> Access violation are logged until the bit is cleared.
	0	RWIC	0	Write Access Violation Interrupt register Asserted when a write access causes the Interconnect to return a DECERR response. Cleared by setting the bit to 1. <i>Note:</i> Access violation are logged until the bit is cleared.

### 14.2.2.3 CSR Read Access Violation Log Registers

The CSR read access violation log settings are valid only when an associated read interrupt register is set. Read this set of registers until the validity bit is cleared.

**Table 203. CSR Read Access Violation Log Registers**

Offset	Bits	Attribute	Default	Description
0x100	31:13	Reserved.		
	12:11	R0	0	Offending Read cycle burst type: Specifies the burst type of the initiating cycle that causes the access violation.
	10:7	R0	0	Offending Read cycle burst length: Specifies the burst length of the initiating cycle that causes the access violation.
	6:4	R0	0	Offending Read cycle burst size: Specifies the burst size of the initiating cycle that causes the access violation.
	3:1	R0	0	Offending Read cycle PROT: Specifies the PROT of the initiating cycle that causes the access violation.
	0	R0	0	Read cycle log valid: Specifies the validity of the read access violation log. This bit is cleared when the interrupt register is cleared.
0x104	31:0	R0	0	Offending read cycle ID: Master ID for the cycle that causes the access violation.
0x108	31:0	R0	0	Offending read cycle target address: Target address for the cycle that causes the access violation (lower 32-bit).
0x10C	31:0	R0	0	Offending read cycle target address: Target address for the cycle that causes the access violation (upper 32-bit). Valid only if widest address in system is larger than 32 bits. <i>Note:</i> When this register is read, the current read access violation log is recovered from FIFO.

### 14.2.2.4 CSR Write Access Violation Log Registers

The CSR write access violation log settings are valid only when an associated write interrupt register is set. Read this set of registers until the validity bit is cleared.

**Table 204. CSR Write Access Violation Log**

Offset	Bits	Attribute	Default	Description
0x190	31:13	Reserved.		
	12:11	R0	0	Offending write cycle burst type: Specifies the burst type of the initiating cycle that causes the access violation.
	10:7	R0	0	Offending write cycle burst length: Specifies the burst length of the initiating cycle that causes the access violation.
<i>continued...</i>				



Offset	Bits	Attribute	Default	Description
	6:4	R0	0	Offending write cycle burst size: Specifies the burst size of the initiating cycle that causes the access violation.
	3:1	R0	0	Offending write cycle PROT: Specifies the PROT of the initiating cycle that causes the access violation.
	0	R0	0	Write cycle log valid: Specifies whether the log for the transaction is valid. This bit is cleared when the interrupt register is cleared.
0x194	31:0	R0	0	Offending write cycle ID: Master ID for the cycle that causes the access violation.
0x198	31:0	R0	0	Offending write cycle target address: Write target address for the cycle that causes the access violation (lower 32-bit).
0x19C	31:0	R0	0	Offending write cycle target address: Write target address for the cycle that causes the access violation (upper 32-bit). Valid only if widest address in system is larger than 32 bits.
0x1A0	31:0	R0	0	Offending write cycle first write data: First 32 bits of the write data for the write cycle that causes the access violation. <i>Note:</i> When this register is read, the current write access violation log is recovered from FIFO, when the data width is 32 bits.
0x1A4	31:0	R0	0	Offending write cycle first write data: Bits [63:32] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 32 bits.
0x1A8	31:0	R0	0	Offending write cycle first write data: Bits [95:64] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 64 bits.
0x1AC	31:0	R0	0	Offending write cycle first write data: The first bits [127:96] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 64 bits. <i>Note:</i> When this register is read, the current write access violation log is recovered from FIFO.

### 14.2.3 Designating a Default Slave in the System Contents Tab

You can designate any slave in your Platform Designer system as the error response default slave. The default slave you designate provides an error response service for masters that attempt access to an undefined memory region.

1. In your Platform Designer system, in the **System Contents** tab, right-click the header and turn on **Show Default Slave Column**.
2. Select the slave that you want to designate as the default slave, and then click the checkbox for the slave in the **Default Slave** column.
3. In the **System Contents** tab, in the **Connections** column, connect the designated default slave to one or more masters.

#### Related Links

[Specify a Default Slave in a Platform Designer System](#) on page 374



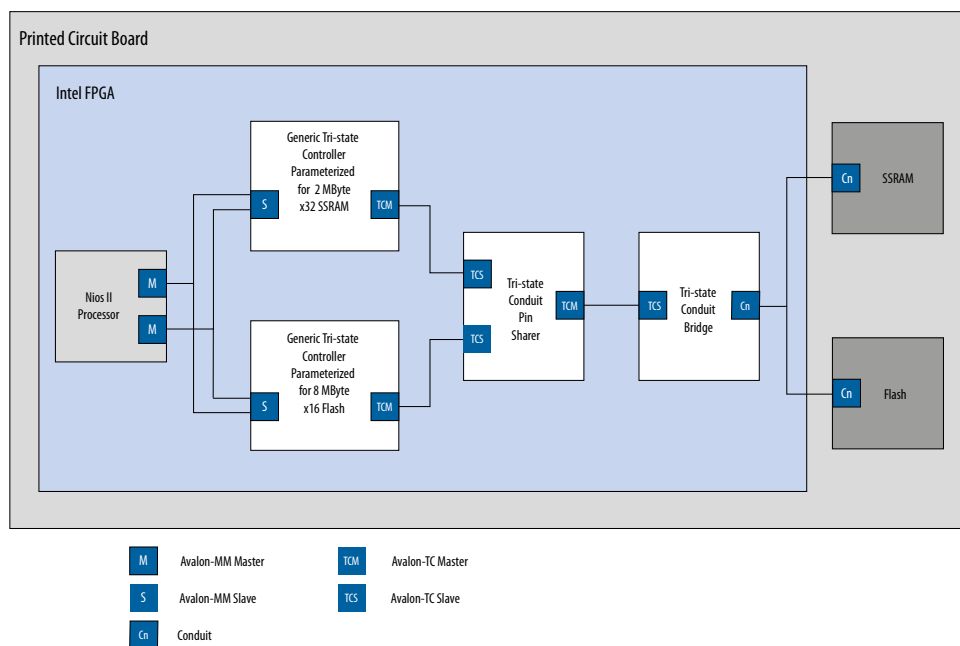
### 14.3 Tri-State Components

The tri-state interface type allows you to design Platform Designer subsystems that connect to tri-state devices on your PCB. You can use tri-state components to implement pin sharing, convert between unidirectional and bidirectional signals, and create tri-state controllers for devices whose interfaces can be described using the tri-state signal types.

#### Example 104. Tri-State Conduit System to Control Off-Chip SRAM and Flash Devices

In this example, there are two generic Tri-State Conduit Controllers. The first is customized to control a flash memory. The second is customized to control an off-chip SSRAM. The Tri-State Conduit Pin Sharer multiplexes between these two controllers, and the Tri-State Conduit Bridge converts between an on-chip encoding of tri-state signals and true bidirectional signals. By default, the Tri-State Conduit Pin Sharer and Tri-State Conduit Bridge present byte addresses. Typically, each address location contains more than one byte of data.

Figure 301. Tri-State Conduit System to Control Off-Chip SRAM and Flash Devices



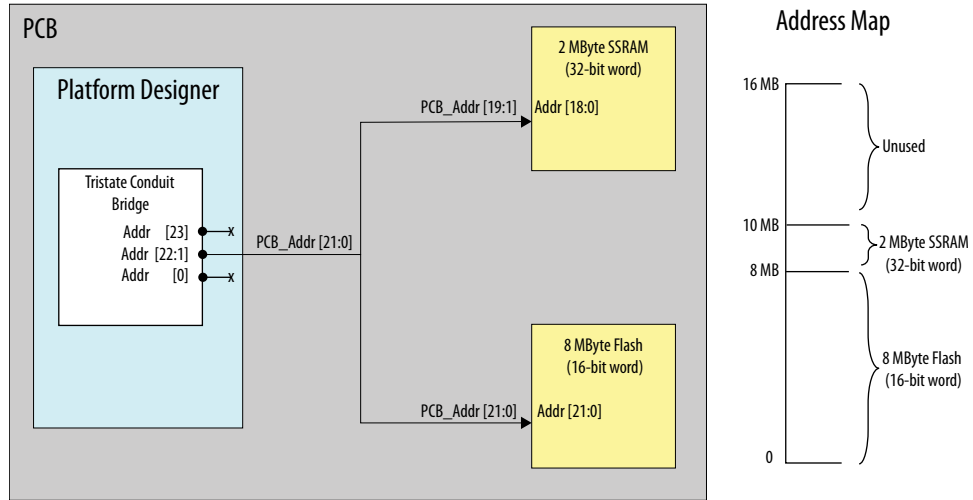
#### Address Connections from Platform Designer System to PCB

The flash device operates on 16-bit words and must ignore the least-significant bit of the Avalon-MM address. The figure shows `addr[0]` as not connected. The SSRAM memory operates on 32-bit words and must ignore the two low-order memory bits. Because neither device requires a byte address, `addr[0]` is not routed on the PCB.

The flash device responds to address range 0 MB to 8 MB-1. The SSRAM responds to address range 8 MB to 10 MB-1. The PCB schematic for the PCB connects `addr[21:0]` to `addr[18:0]` of the SSRAM device because the SSRAM responds to

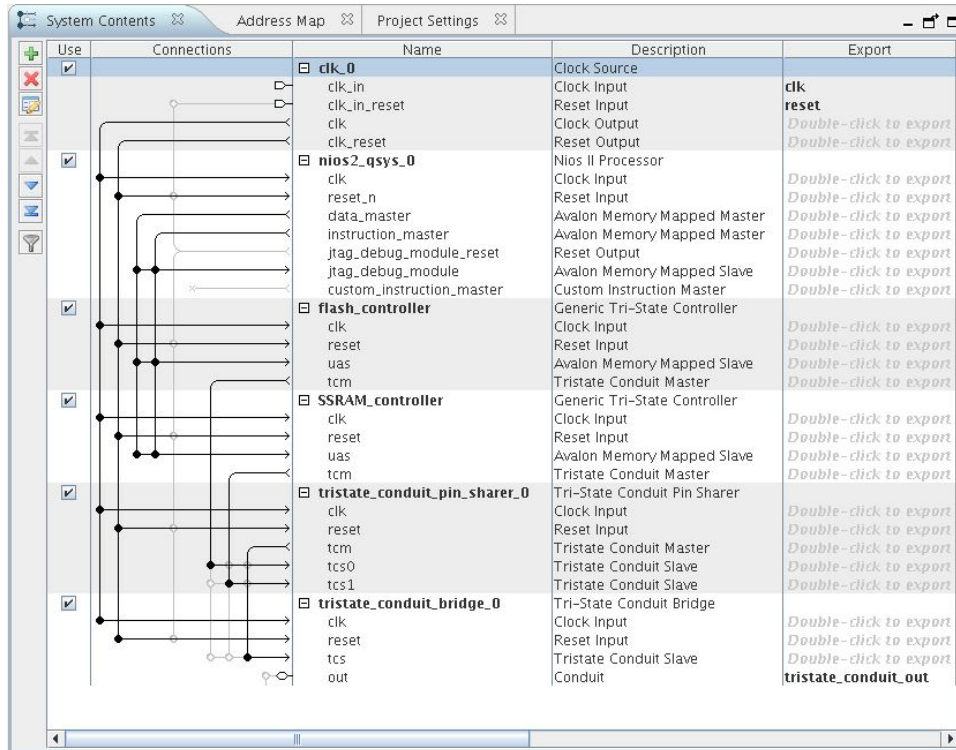
32-bit word address. The 8 MB flash device accesses 16-bit words; consequently, the schematic does not connect `addr[0]`. The `chipselct` signals select between the two devices.

**Figure 302. Address Connections from Platform Designer System to PCB**



**Note:** If you create a custom tri-state conduit master with word aligned addresses, the Tri-state Conduit Pin Sharer does not change or align the address signals.

**Figure 303. Tri-State Conduit System in Platform Designer**







### Related Links

- [Avalon Tri-State Conduit Components User Guide](#)
- [Avalon Interface Specifications](#)

## 14.3.1 Generic Tri-State Controller

The Generic Tri-State Controller provides a template for a controller. You can customize the tri-state controller with various parameters to reflect the behavior of an off-chip device. The following types of parameters are available for the tri-state controller:

- Width of the address and data signals
- Read and write wait times
- Bus-turnaround time
- Data hold time

*Note:* In calculating delays, the Generic Tri-State Controller chooses the larger of the bus-turnaround time and read latency. Turnaround time is measured from the time that a command is accepted, not from the time that the previous read returned data.

The Generic Tri-State Controller includes the following interfaces:

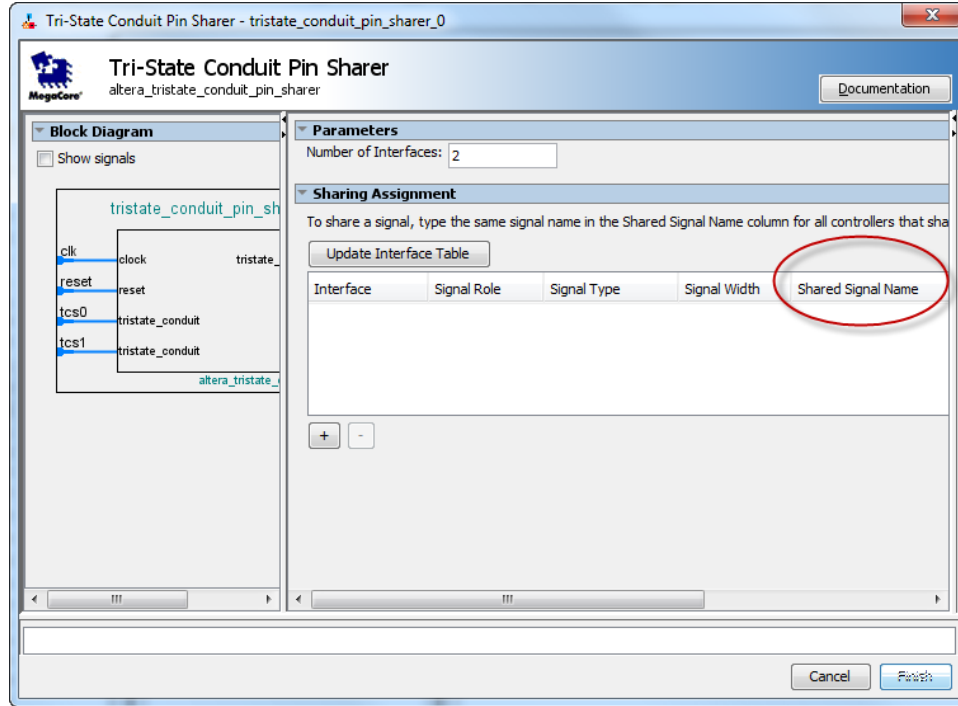
- **Memory-mapped slave interface**—This interface connects to a memory-mapped master, such as a processor.
- **Tristate Conduit Master interface**—The tri-state master interface usually connects to the tri-state conduit slave interface of the tri-state conduit pin sharer.
- **Clock sink**—The component's clock reference. You must connect this interface to a clock source.
- **Reset sink**—This interface connects to a reset source interface.

## 14.3.2 Tri-State Conduit Pin Sharer

The Tri-state Conduit Pin Sharer multiplexes between the signals of the connected tri-state controllers. You connect all signals from the tri-state controllers to the Tri-state Conduit Pin Sharer and use the parameter editor to specify the signals that are shared.

**Figure 304. Tri-State Conduit Pin Sharer Parameter Editor**

The parameter editor includes a **Shared Signal Name** column. If the widths of shared signals differ, the signals are aligned on their 0<sup>th</sup> bit and the higher-order pins are driven to 0 whenever the smaller signal has control of the bus. Unshared signals always propagate through the pin sharer. The tri-state conduit pin sharer uses the round-robin arbiter to select between tri-state conduit controllers.



**Note:** All tri-state conduit components connected to a pin sharer must be in the same clock domain.

**Related Links**

[Avalon-ST Round Robin Scheduler](#) on page 970

**14.3.3 Tri-State Conduit Bridge**

The Tri-State Conduit Bridge instantiates bidirectional signals for each tri-state signal while passing all other signals straight through the component. The Tri-State Conduit Bridge registers all outgoing and incoming signals, which adds two cycles of latency for a read request. You must account for this additional pipelining when designing a custom controller. During reset, all outputs are placed in a high-impedance state. Outputs are enabled in the first clock cycle after reset is deasserted, and the output signals are then bidirectional.

**14.4 Test Pattern Generator and Checker Cores**

The test pattern generator inserts different error conditions, and the test pattern checker reports these error conditions to the control interface, each via an Avalon Memory-Mapped (Avalon-MM) slave.



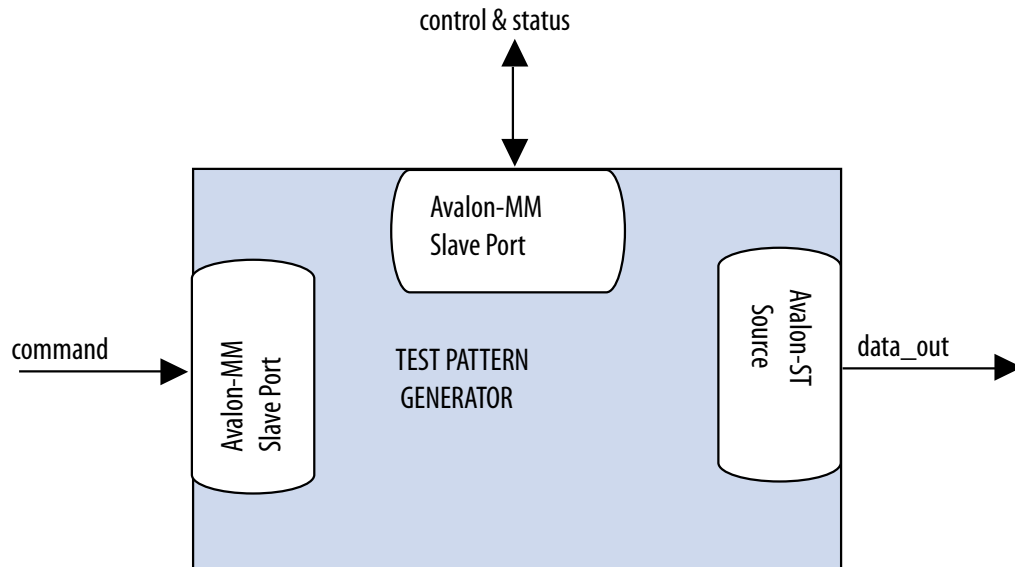
The data generation and monitoring solution for Avalon-ST consists of two components: a test pattern generator core that generates data, and sends it out on an Avalon-ST data interface, and a test pattern checker core that receives the same data and verifies it. Optionally, the data can be formatted as packets, with accompanying `start_of_packet` and `end_of_packet` signals.

The **Throttle Seed** is the starting value for the throttle control random number generator. Intel recommends a unique value for each instance of the test pattern generator and checker cores in a system.

### 14.4.1 Test Pattern Generator

Figure 305. Test Pattern Generator Core

The test pattern generator core accepts commands to generate data via an Avalon-MM command interface, and drives the generated data to an Avalon-ST data interface. You can parameterize most aspects of the Avalon-ST data interface, such as the number of error bits and data signal width, thus allowing you to test components with different interfaces.



The data pattern is calculated as:  $Symbol\ Value = Symbol\ Position\ in\ Packet \oplus Data\ Error\ Mask$ . Data that is not organized in packets is a single stream with no beginning or end. The test pattern generator has a throttle register that is set via the Avalon-MM control interface. The test pattern generator uses the value of the throttle register in conjunction with a pseudo-random number generator to throttle the data generation rate.

#### 14.4.1.1 Test Pattern Generator Command Interface

The command interface for the Test Pattern Generator is a 32-bit Avalon-MM write slave that accepts data generation commands. It is connected to a 16-element deep FIFO, thus allowing a master peripheral to drive commands into the test pattern generator.

The command interface maps to the following registers: `cmd_lo` and `cmd_hi`. The command is pushed into the FIFO when the register `cmd_lo` (address 0) is addressed. When the FIFO is full, the command interface asserts the `waitrequest` signal. You can create errors by writing to the register `cmd_hi` (address 1). The errors are cleared when 0 is written to this register, or its respective fields.

#### 14.4.1.2 Test Pattern Generator Control and Status Interface

The control and status interface of the Test Pattern Generator is a 32-bit Avalon-MM slave that allows you to enable or disable the data generation, as well as set the throttle. This interface also provides generation-time information, such as the number of channels and whether data packets are supported.

#### 14.4.1.3 Test Pattern Generator Output Interface

The output interface of the Test Pattern Generator is an Avalon-ST interface that optionally supports data packets. You can configure the output interface to align with your system requirements. Depending on the incoming stream of commands, the output data may contain interleaved packet fragments for different channels. To keep track of the current symbol's position within each packet, the test pattern generator maintains an internal state for each channel.

You can configure the output interface of the test pattern generator with the following parameters:

- **Number of Channels**—Number of channels that the test pattern generator supports. Valid values are 1 to 256.
- **Data Bits Per Symbol**—Bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—Number of symbols (words) that are transferred per beat. Valid values are 1 to 256.
- **Include Packet Support**—Indicates whether packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Error Signal Width (bits)**—Width of the error signal on the output interface. Valid values are 0 to 31. A value of 0 indicates that the error signal is not in use.

*Note:* If you change only bits per symbol, and do not change the data width, errors are generated.

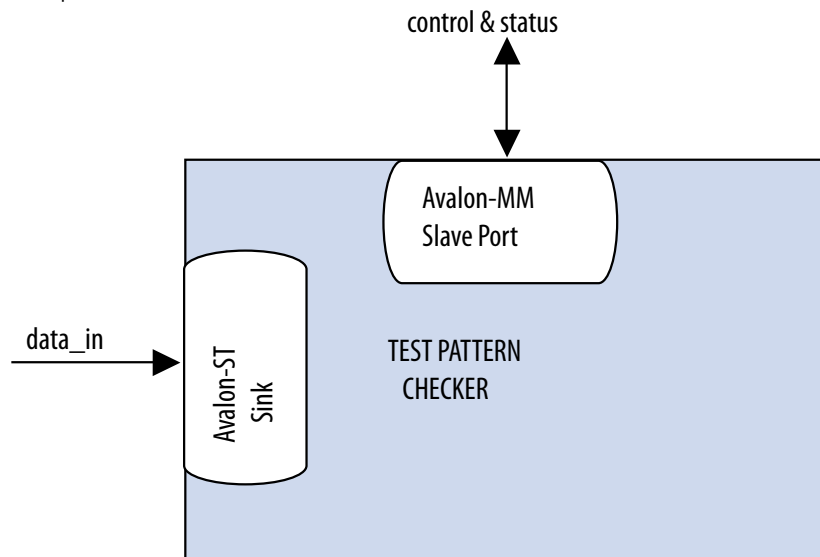
#### 14.4.1.4 Test Pattern Generator Functional Parameter

The Test Pattern Generator functional parameter allows you to configure the test pattern generator as a whole system.

## 14.4.2 Test Pattern Checker

**Figure 306. Test Pattern Checker**

The test pattern checker core accepts data via an Avalon-ST interface and verifies it against the same predetermined pattern that the test pattern generator uses to produce the data. The test pattern checker core reports any exceptions to the control interface. You can parameterize most aspects of the test pattern checker's Avalon-ST interface such as the number of error bits and the data signal width. This enables the ability to test components with different interfaces. The test pattern checker has a throttle register that is set via the Avalon-MM control interface. The value of the throttle register controls the rate at which data is accepted.



The test pattern checker detects exceptions and reports them to the control interface via a 32-element deep internal FIFO. Possible exceptions are data error, missing start-of-packet (SOP), missing end-of-packet (EOP), and signaled error.

As each exception occurs, an exception descriptor is pushed into the FIFO. If the same exception occurs more than once consecutively, only one exception descriptor is pushed into the FIFO. All exceptions are ignored when the FIFO is full. Exception descriptors are deleted from the FIFO after they are read by the control and status interface.

### 14.4.2.1 Test Pattern Checker Input Interface

The Test Pattern Checker input interface is an Avalon-ST interface that optionally supports data packets. You can configure the input interface to align with your system requirements. Incoming data may contain interleaved packet fragments. To keep track of the current symbol's position, the test pattern checker maintains an internal state for each channel.

### 14.4.2.2 Test Pattern Checker Control and Status Interface

The Test Pattern Checker control and status interface is a 32-bit Avalon-MM slave that allows you to enable or disable data acceptance, as well as set the throttle. This interface provides generation-time information, such as the number of channels and whether the test pattern checker supports data packets. The control and status interface also provides information on the exceptions detected by the test pattern checker. The interface obtains this information by reading from the exception FIFO.

### 14.4.2.3 Test Pattern Checker Functional Parameter

The Test Pattern Checker functional parameter allows you to configure the test pattern checker as a whole system.

### 14.4.2.4 Test Pattern Checker Input Parameters

You can configure the input interface of the test pattern checker using the following parameters:

- **Data Bits Per Symbol**—Bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—Number of symbols (words) that are transferred per beat. Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Number of Channels**—Number of channels that the test pattern checker supports. Valid values are 1 to 256.
- **Error Signal Width (bits)**—Width of the `error` signal on the input interface. Valid values are 0 to 31. A value of 0 indicates that the `error` signal is not in use.

*Note:* If you change only bits per symbol, and do not change the data width, errors are generated.

## 14.4.3 Software Programming Model for the Test Pattern Generator and Checker Cores

The HAL system library support, software files, and register maps describe the software programming model for the test pattern generator and checker cores.

### 14.4.3.1 HAL System Library Support

For Nios II processor users, Intel provides HAL system library drivers that allow you to initialize and access the test pattern generator and checker cores. Intel recommends you use the provided drivers to access the cores instead of accessing the registers directly.

For Nios II IDE users, copy the provided drivers from the following installation folders to your software application directory:

- `<IP installation directory>/ip/sopc_builder_ip/altera_Avalon_data_source/HAL`
- `<IP installation directory>/ip/sopc_builder_ip/altera_Avalon_data_sink/HAL`

*Note:* This instruction does not apply if you use the Nios II command-line tools.

### 14.4.3.2 Test Pattern Generator and Test Pattern Checker Core Files

The following files define the low-level access to the hardware, and provide the routines for the HAL device drivers.



**Note:** Do not modify the test pattern generator or test pattern checker core files.

- Test pattern generator core files:
  - **data\_source\_regs.h**—Header file that defines the test pattern generator's register maps.
  - **data\_source\_util.h, data\_source\_util.c**—Header and source code for the functions and variables required to integrate the driver into the HAL system library.
- Test pattern checker core files:
  - **data\_sink\_regs.h**—Header file that defines the core's register maps.
  - **data\_sink\_util.h, data\_sink\_util.c**—Header and source code for the functions and variables required to integrate the driver into the HAL system library.

### 14.4.3.3 Register Maps for the Test Pattern Generator and Test Pattern Checker Cores

#### 14.4.3.3.1 Test Pattern Generator Control and Status Registers

**Table 205. Test Pattern Generator Control and Status Register Map**

Shows the offset for the test pattern generator control and status registers. Each register is 32-bits wide.

Offset	Register Name
base + 0	status
base + 1	control
base + 2	fill

**Table 206. Test Pattern Generator Status Register Bits**

Bits	Name	Access	Description
[15:0]	ID	RO	A constant value of 0x64.
[23:16]	NUMCHANNELS	RO	The configured number of channels.
[30:24]	NUMSYMBOLS	RO	The configured number of symbols per beat.
[31]	SUPPORTPACKETS	RO	A value of 1 indicates data packet support.

**Table 207. Test Pattern Generator Control Register Bits**

Bits	Name	Access	Description
[0]	ENABLE	RW	Setting this bit to 1 enables the test pattern generator core.
[7:1]	Reserved		
[16:8]	THROTTLE	RW	Specifies the throttle value which can be between 0–256, inclusively. The test pattern generator uses this value in conjunction with a pseudo-random number generator to throttle the data generation rate.
[17]	SOFT RESET	RW	When this bit is set to 1, all internal counters and statistics are reset. Write 0 to this bit to exit reset.
[31:18]	Reserved		

**Table 208. Test Pattern Generator Fill Register Bits**

Bits	Name	Access	Description
[0]	BUSY	RO	A value of 1 indicates that data transmission is in progress, or that there is at least one command in the command queue.
[6:1]	Reserved		
[15:7]	FILL	RO	The number of commands currently in the command FIFO.
[31:16]	Reserved		

#### 14.4.3.3.2 Test Pattern Generator Command Registers

**Table 209. Test Pattern Generator Command Register Map**

Shows the offset for the command registers. Each register is 32-bits wide.

Offset	Register Name
base + 0	cmd_lo
base + 1	cmd_hi

The `cmd_lo` is pushed into the FIFO only when the `cmd_lo` register is addressed.

**Table 210. cmd\_lo Register Bits**

Bits	Name	Access	Description
[15:0]	SIZE	RW	The segment size in symbols. Except for the last segment in a packet, the size of all segments must be a multiple of the configured number of symbols per beat. If this condition is not met, the test pattern generator core inserts additional symbols to the segment to ensure the condition is fulfilled.
[29:16]	CHANNEL	RW	The channel to send the segment on. If the <code>channel</code> signal is less than 14 bits wide, the test pattern generator uses the low order bits of this register to drive the signal.
[30]	SOP	RW	Set this bit to 1 when sending the first segment in a packet. This bit is ignored when data packets are not supported.
[31]	EOP	RW	Set this bit to 1 when sending the last segment in a packet. This bit is ignored when data packets are not supported.

**Table 211. cmd\_hi Register Bits**

Bits	Name	Access	Description
[15:0]	SIGNALLED ERROR	RW	Specifies the value to drive the <code>error</code> signal. A non-zero value creates a signaled error.
[23:16]	DATA ERROR	RW	The output data is XORED with the contents of this register to create data errors. To stop creating data errors, set this register to 0.
[24]	SUPPRESS SOP	RW	Set this bit to 1 to suppress the assertion of the <code>startofpacket</code> signal when the first segment in a packet is sent.
[25]	SUPPRESS EOP	RW	Set this bit to 1 to suppress the assertion of the <code>endofpacket</code> signal when the last segment in a packet is sent.





### 14.4.3.3.3 Test Pattern Checker Control and Status Registers

**Table 212. Test Pattern Checker Control and Status Register Map**

Shows the offset for the control and status registers. Each register is 32 bits wide.

Offset	Register Name
base + 0	status
base + 1	control
base + 2	Reserved
base + 3	
base + 4	
base + 5	exception_descriptor
base + 6	indirect_select
base + 7	indirect_count

**Table 213. Test Pattern Checker Status Register Bits**

Bit(s)	Name	Access	Description
[15:0]	ID	RO	Contains a constant value of 0x65.
[23:16]	NUMCHANNELS	RO	The configured number of channels.
[30:24]	NUMSYMBOLS	RO	The configured number of symbols per beat.
[31]	SUPPORTPACKETS	RO	A value of 1 indicates packet support.

**Table 214. Test Pattern Checker Control Register Bits**

Bit(s)	Name	Access	Description
[0]	ENABLE	RW	Setting this bit to 1 enables the test pattern checker.
[7:1]	Reserved		
[16:8]	THROTTLE	RW	Specifies the throttle value which can be between 0–256, inclusively. Platform Designer uses this value in conjunction with a pseudo-random number generator to throttle the data generation rate. Setting THROTTLE to 0 stops the test pattern generator core. Setting it to 256 causes the test pattern generator core to run at full throttle. Values between 0–256 result in a data rate proportional to the throttle value.
[17]	SOFT RESET	RW	When this bit is set to 1, all internal counters and statistics are reset. Write 0 to this bit to exit reset.
[31:18]	Reserved		

If there is no exception, reading the `exception_descriptor` register bit register returns 0.

**Table 215. `exception_descriptor` Register Bits**

Bit(s)	Name	Access	Description
[0]	DATA_ERROR	RO	A value of 1 indicates that an error is detected in the incoming data.
[1]	MISSINGSOP	RO	A value of 1 indicates missing start-of-packet.
<i>continued...</i>			

Bit(s)	Name	Access	Description
[2]	MISSINGEOP	RO	A value of 1 indicates missing end-of-packet.
[7:3]	Reserved		
[15:8]	SIGNALLED ERROR	RO	The value of the error signal.
[23:16]	Reserved		
[31:24]	CHANNEL	RO	The channel on which the exception was detected.

**Table 216.** `indirect_select` Register Bits

Bit	Bits Name	Access	Description
[7:0]	INDIRECT CHANNEL	RW	Specifies the channel number that applies to the INDIRECT PACKET COUNT, INDIRECT SYMBOL COUNT, and INDIRECT ERROR COUNT registers.
[15:8]	Reserved		
[31:16]	INDIRECT ERROR	RO	The number of data errors that occurred on the channel specified by INDIRECT CHANNEL.

**Table 217.** `indirect_count` Register Bits

Bit	Bits Name	Access	Description
[15:0]	INDIRECT PACKET COUNT	RO	The number of data packets received on the channel specified by INDIRECT CHANNEL.
[31:16]	INDIRECT SYMBOL COUNT	RO	The number of symbols received on the channel specified by INDIRECT CHANNEL.

### 14.4.4 Test Pattern Generator API

The following subsections describe application programming interface (API) for the test pattern generator.

*Note:* API functions are currently not available from the interrupt service routine (ISR).

[data\\_source\\_reset\(\)](#) on page 955

[data\\_source\\_init\(\)](#) on page 955

[data\\_source\\_get\\_id\(\)](#) on page 955

[data\\_source\\_get\\_supports\\_packets\(\)](#) on page 956

[data\\_source\\_get\\_num\\_channels\(\)](#) on page 956

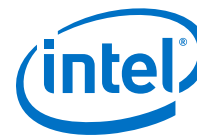
[data\\_source\\_get\\_symbols\\_per\\_cycle\(\)](#) on page 956

[data\\_source\\_get\\_enable\(\)](#) on page 956

[data\\_source\\_set\\_enable\(\)](#) on page 957

[data\\_source\\_get\\_throttle\(\)](#) on page 957

[data\\_source\\_set\\_throttle\(\)](#) on page 957



[data\\_source\\_is\\_busy\(\)](#) on page 958

[data\\_source\\_fill\\_level\(\)](#) on page 958

[data\\_source\\_send\\_data\(\)](#) on page 958

#### 14.4.4.1 data\_source\_reset()

**Table 218.** data\_source\_reset()

Information Type	Description
<b>Prototype</b>	<code>void data_source_reset(alt_u32 base);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	<code>void</code>
<b>Description</b>	Resets the test pattern generator core including all internal counters and FIFOs. The control and status registers are not reset by this function.

#### 14.4.4.2 data\_source\_init()

**Table 219.** data\_source\_init()

Information Type	Description
<b>Prototype</b>	<code>int data_source_init(alt_u32 base, alt_u32 command_base);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave. <code>command_base</code> —Base address of the command slave.
<b>Returns</b>	1—Initialization is successful. 0—Initialization is unsuccessful.
<b>Description</b>	Performs the following operations to initialize the test pattern generator core: <ul style="list-style-type: none"> <li>Resets and disables the test pattern generator core.</li> <li>Sets the maximum throttle.</li> <li>Clears all inserted errors.</li> </ul>

#### 14.4.4.3 data\_source\_get\_id()

**Table 220.** data\_source\_get\_id()

Information Type	Description
<b>Prototype</b>	<code>int data_source_get_id(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	Test pattern generator core identifier.
<b>Description</b>	Retrieves the test pattern generator core's identifier.

#### 14.4.4.4 data\_source\_get\_supports\_packets()

Table 221. data\_source\_get\_supports\_packets()

Information Type	Description
<b>Prototype</b>	<code>int data_source_init(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	1—Data packets are supported. 0—Data packets are not supported.
<b>Description</b>	Checks if the test pattern generator core supports data packets.

#### 14.4.4.5 data\_source\_get\_num\_channels()

Table 222. data\_source\_get\_num\_channels()

Description	Description
<b>Prototype</b>	<code>int data_source_get_num_channels(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	Number of channels supported.
<b>Description</b>	Retrieves the number of channels supported by the test pattern generator core.

#### 14.4.4.6 data\_source\_get\_symbols\_per\_cycle()

Table 223. data\_source\_get\_symbols\_per\_cycle()

Description	Description
<b>Prototype</b>	<code>int data_source_get_symbols(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	Number of symbols transferred in a beat.
<b>Description</b>	Retrieves the number of symbols transferred by the test pattern generator core in each beat.

#### 14.4.4.7 data\_source\_get\_enable()

Table 224. data\_source\_get\_enable()

Information Type	Description
<b>Prototype</b>	<code>int data_source_get_enable(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<i>continued...</i>	



Information Type	Description
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	Value of the ENABLE bit.
<b>Description</b>	Retrieves the value of the ENABLE bit.

#### 14.4.4.8 data\_source\_set\_enable()

Table 225. data\_source\_set\_enable()

Information Type	Description
<b>Prototype</b>	<code>void data_source_set_enable(alt_u32 base, alt_u32 value);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave. value— ENABLE bit set to the value of this parameter.
<b>Returns</b>	void
<b>Description</b>	Enables or disables the test pattern generator core. When disabled, the test pattern generator core stops data transmission but continues to accept commands and stores them in the FIFO

#### 14.4.4.9 data\_source\_get\_throttle()

Table 226. data\_source\_get\_throttle()

Information Type	Description
<b>Prototype</b>	<code>int data_source_get_throttle(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	Throttle value.
<b>Description</b>	Retrieves the current throttle value.

#### 14.4.4.10 data\_source\_set\_throttle()

Table 227. data\_source\_set\_throttle()

Information Type	Description
<b>Prototype</b>	<code>void data_source_set_throttle(alt_u32 base, alt_u32 value);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<i>continued...</i>	

Information Type	Description
	value—Throttle value.
<b>Returns</b>	void
<b>Description</b>	Sets the throttle value, which can be between 0–256 inclusively. The throttle value, when divided by 256 yields the rate at which the test pattern generator sends data.

#### 14.4.4.11 data\_source\_is\_busy()

Table 228. data\_source\_is\_busy()

Information Type	Description
<b>Prototype</b>	<code>int data_source_is_busy(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	1—Test pattern generator core is busy. 0—Test pattern generator core is not busy.
<b>Description</b>	Checks if the test pattern generator is busy. The test pattern generator core is busy when it is sending data or has data in the command FIFO to be sent.

#### 14.4.4.12 data\_source\_fill\_level()

Table 229. data\_source\_fill\_level()

Information Type	Description
<b>Prototype</b>	<code>int data_source_fill_level(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	Number of commands in the command FIFO.
<b>Description</b>	Retrieves the number of commands currently in the command FIFO.

#### 14.4.4.13 data\_source\_send\_data()

Table 230. data\_source\_send\_data()

Information Type	Description
<b>Prototype</b>	<code>int data_source_send_data(alt_u32 cmd_base, alt_u16 channel, alt_u16 size, alt_u32 flags, alt_u16 error, alt_u8 data_error_mask);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_source_util.h &gt;</code>
<b>Parameters</b>	cmd_base—Base address of the command slave. channel—Channel to send the data.

*continued...*



Information Type	Description
	<p><code>size</code>—Data size.</p> <p><code>flags</code> —Specifies whether to send or suppress SOP and EOP signals. Valid values are <code>DATA_SOURCE_SEND_SOP</code>, <code>DATA_SOURCE_SEND_EOP</code>, <code>DATA_SOURCE_SEND_SUPPRESS_SOP</code> and <code>DATA_SOURCE_SEND_SUPPRESS_EOP</code>.</p> <p><code>error</code>—Value asserted on the <code>error</code> signal on the output interface.</p> <p><code>data_error_mask</code>—Parameter and the data are XORed together to produce erroneous data.</p>
<b>Returns</b>	Returns 1.
<b>Description</b>	<p>Sends a data fragment to the specified channel. If data packets are supported, applications must ensure consistent usage of SOP and EOP in each channel. Except for the last segment in a packet, the length of each segment is a multiple of the data width.</p> <p>If data packets are not supported, applications must ensure that there are no SOP and EOP indicators in the data. The length of each segment in a packet is a multiple of the data width.</p>

### 14.4.5 Test Pattern Checker API

The following subsections describe API for the test pattern checker core. The API functions are currently not available from the ISR.

- [data\\_sink\\_reset\(\)](#) on page 960
- [data\\_sink\\_init\(\)](#) on page 960
- [data\\_sink\\_get\\_id\(\)](#) on page 960
- [data\\_sink\\_get\\_supports\\_packets\(\)](#) on page 961
- [data\\_sink\\_get\\_num\\_channels\(\)](#) on page 961
- [data\\_sink\\_get\\_symbols\\_per\\_cycle\(\)](#) on page 961
- [data\\_sink\\_get\\_enable\(\)](#) on page 961
- [data\\_sink\\_set\\_enable\(\)](#) on page 962
- [data\\_sink\\_get\\_throttle\(\)](#) on page 962
- [data\\_sink\\_set\\_throttle\(\)](#) on page 962
- [data\\_sink\\_get\\_packet\\_count\(\)](#) on page 963
- [data\\_sink\\_get\\_error\\_count\(\)](#) on page 963
- [data\\_sink\\_get\\_symbol\\_count\(\)](#) on page 963
- [data\\_sink\\_get\\_exception\(\)](#) on page 964
- [data\\_sink\\_exception\\_is\\_exception\(\)](#) on page 964
- [data\\_sink\\_exception\\_has\\_data\\_error\(\)](#) on page 964
- [data\\_sink\\_exception\\_has\\_missing\\_sop\(\)](#) on page 965
- [data\\_sink\\_exception\\_has\\_missing\\_eop\(\)](#) on page 965
- [data\\_sink\\_exception\\_signalled\\_error\(\)](#) on page 965
- [data\\_sink\\_exception\\_channel\(\)](#) on page 966



### 14.4.5.1 data\_sink\_reset()

Table 231. data\_sink\_reset()

Information Type	Description
<b>Prototype</b>	<code>void data_sink_reset(alt_u32 base);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	void
<b>Description</b>	Resets the test pattern checker core including all internal counters.

### 14.4.5.2 data\_sink\_init()

Table 232. data\_sink\_init()

Information Type	Description
<b>Prototype</b>	<code>int data_source_init(alt_u32 base);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	1—Initialization is successful. 0—Initialization is unsuccessful.
<b>Description</b>	Performs the following operations to initialize the test pattern checker core: <ul style="list-style-type: none"><li>Resets and disables the test pattern checker core.</li><li>Sets the throttle to the maximum value.</li></ul>

### 14.4.5.3 data\_sink\_get\_id()

Table 233. data\_sink\_get\_id()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_get_id(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	<code>base</code> —Base address of the control and status slave.
<b>Returns</b>	Test pattern checker core identifier.
<b>Description</b>	Retrieves the test pattern checker core's identifier.





#### 14.4.5.4 data\_sink\_get\_supports\_packets()

Table 234. data\_sink\_get\_supports\_packets()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_init(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h&gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	1—Data packets are supported. 0—Data packets are not supported.
<b>Description</b>	Checks if the test pattern checker core supports data packets.

#### 14.4.5.5 data\_sink\_get\_num\_channels()

Table 235. data\_sink\_get\_num\_channels()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_get_num_channels(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h&gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	Number of channels supported.
<b>Description</b>	Retrieves the number of channels supported by the test pattern checker core.

#### 14.4.5.6 data\_sink\_get\_symbols\_per\_cycle()

Table 236. data\_sink\_get\_symbols\_per\_cycle()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_get_symbols(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h&gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	Number of symbols received in a beat.
<b>Description</b>	Retrieves the number of symbols received by the test pattern checker core in each beat.

#### 14.4.5.7 data\_sink\_get\_enable()

Table 237. data\_sink\_get\_enable()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_get_enable(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<i>continued...</i>	



Information Type	Description
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	Value of the ENABLE bit.
<b>Description</b>	Retrieves the value of the ENABLE bit.

#### 14.4.5.8 data\_sink\_set\_enable()

Table 238. data\_sink\_set\_enable()

Information Type	Description
<b>Prototype</b>	<code>void data_sink_set_enable(alt_u32 base, alt_u32 value);</code>
<b>Thread-safe</b>	No
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave. value—ENABLE bit is set to the value of the parameter.
<b>Returns</b>	void
<b>Description</b>	Enables the test pattern checker core.

#### 14.4.5.9 data\_sink\_get\_throttle()

Table 239. data\_sink\_get\_throttle()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_get_throttle(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	Throttle value.
<b>Description</b>	Retrieves the throttle value.

#### 14.4.5.10 data\_sink\_set\_throttle()

Table 240. data\_sink\_set\_throttle()

Information Type	Description
<b>Prototype</b>	<code>void data_sink_set_throttle(alt_u32 base, alt_u32 value);</code>
<b>Thread-safe</b>	No
<b>Include:</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<i>continued...</i>	



Information Type	Description
	value—Throttle value.
<b>Returns</b>	void
<b>Description</b>	Sets the throttle value, which can be between 0–256 inclusively. The throttle value, when divided by 256 yields the rate at which the test pattern checker receives data.

#### 14.4.5.11 data\_sink\_get\_packet\_count()

Table 241. data\_sink\_get\_packet\_count()

Information Type	Description
<b>Prototype</b>	int data_sink_get_packet_count(alt_u32 base, alt_u32 channel);
<b>Thread-safe</b>	No
<b>Include</b>	<data_sink_util.h >
<b>Parameters</b>	base—Base address of the control and status slave. channel—Channel number.
<b>Returns</b>	Number of data packets received on the channel.
<b>Description</b>	Retrieves the number of data packets received on a channel.

#### 14.4.5.12 data\_sink\_get\_error\_count()

Table 242. data\_sink\_get\_error\_count()

Information Type	Description
<b>Prototype</b>	int data_sink_get_error_count(alt_u32 base, alt_u32 channel);
<b>Thread-safe</b>	No
<b>Include</b>	<data_sink_util.h >
<b>Parameters</b>	base—Base address of the control and status slave. channel—Channel number.
<b>Returns</b>	Number of errors received on the channel.
<b>Description</b>	Retrieves the number of errors received on a channel.

#### 14.4.5.13 data\_sink\_get\_symbol\_count()

Table 243. data\_sink\_get\_symbol\_count()

Information Type	Description
<b>Prototype</b>	int data_sink_get_symbol_count(alt_u32 base, alt_u32 channel);
<b>Thread-safe</b>	No
<b>Include</b>	<data_sink_util.h >
<b>Parameters</b>	base—Base address of the control and status slave.
<i>continued...</i>	

Information Type	Description
	channel—Channel number.
<b>Returns</b>	Number of symbols received on the channel.
<b>Description</b>	Retrieves the number of symbols received on a channel.

#### 14.4.5.14 data\_sink\_get\_exception()

Table 244. data\_sink\_get\_exception()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_get_exception(alt_u32 base);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	base—Base address of the control and status slave.
<b>Returns</b>	First exception descriptor in the exception FIFO. 0—No exception descriptor found in the exception FIFO.
<b>Description</b>	Retrieves the first exception descriptor in the exception FIFO and pops it off the FIFO.

#### 14.4.5.15 data\_sink\_exception\_is\_exception()

Table 245. data\_sink\_exception\_is\_exception()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_exception_is_exception(int exception);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	exception—Exception descriptor
<b>Returns</b>	1—Indicates an exception. 0—No exception.
<b>Description</b>	Checks if an exception descriptor describes a valid exception.

#### 14.4.5.16 data\_sink\_exception\_has\_data\_error()

Table 246. data\_sink\_exception\_has\_data\_error()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_exception_has_data_error(int exception);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	exception—Exception descriptor.
<b>Returns</b>	1—Data has errors. 0—No errors.
<b>Description</b>	Checks if an exception indicates erroneous data.



### 14.4.5.17 data\_sink\_exception\_has\_missing\_sop()

Table 247. data\_sink\_exception\_has\_missing\_sop()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_exception_has_missing_sop(int exception);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	exception—Exception descriptor.
<b>Returns</b>	1—Missing SOP. 0—Other exception types.
<b>Description</b>	Checks if an exception descriptor indicates missing SOP.

### 14.4.5.18 data\_sink\_exception\_has\_missing\_eop()

Table 248. data\_sink\_exception\_has\_missing\_eop()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_exception_has_missing_eop(int exception);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	exception—Exception descriptor.
<b>Returns</b>	1—Missing EOP. 0—Other exception types.
<b>Description</b>	Checks if an exception descriptor indicates missing EOP.

### 14.4.5.19 data\_sink\_exception\_signalled\_error()

Table 249. data\_sink\_exception\_signalled\_error()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_exception_signalled_error(int exception);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	exception—Exception descriptor.
<b>Returns</b>	Signal error value.
<b>Description</b>	Retrieves the value of the signaled error from the exception.

### 14.4.5.20 data\_sink\_exception\_channel()

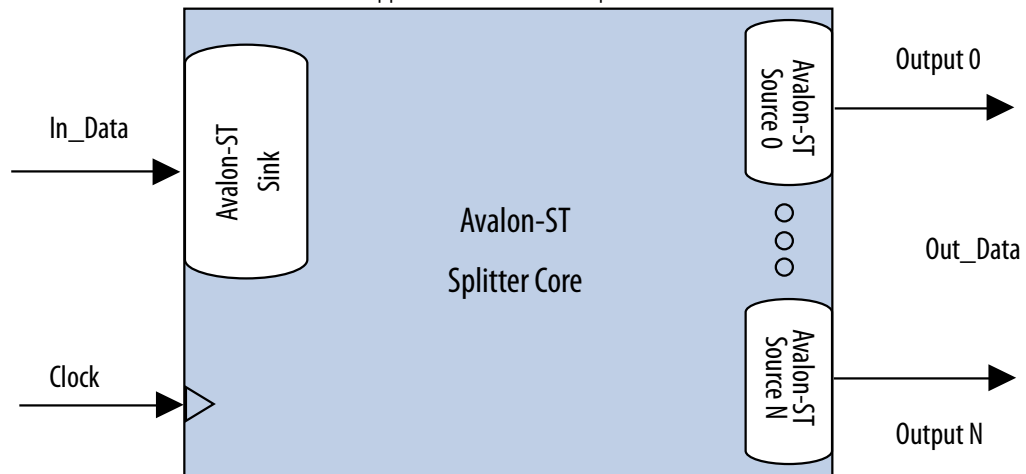
Table 250. data\_sink\_exception\_channel()

Information Type	Description
<b>Prototype</b>	<code>int data_sink_exception_channel(int exception);</code>
<b>Thread-safe</b>	Yes
<b>Include</b>	<code>&lt;data_sink_util.h &gt;</code>
<b>Parameters</b>	<code>exception</code> —Exception descriptor.
<b>Returns</b>	Channel number on which an exception occurred.
<b>Description</b>	Retrieves the channel number on which an exception occurred.

## 14.5 Avalon-ST Splitter Core

Figure 307. Avalon-ST Splitter Core

The Avalon-ST Splitter Core allows you to replicate transactions from an Avalon-ST sink interface to multiple Avalon-ST source interfaces. This core supports from 1 to 16 outputs.



The Avalon-ST Splitter core copies input signals from the input interface to the corresponding output signals of each output interface without altering the size or functionality. This includes all signals except for the `ready` signal. The core includes a clock signal to determine the Avalon-ST interface and clock domain where the core resides. Because the splitter core does not use the `clock` signal internally, latency is not introduced when using this core.

### 14.5.1 Splitter Core Backpressure

The Avalon-ST Splitter core integrates with backpressure by AND-ing the `ready` signals from the output interfaces and sending the result to the input interface. As a result, if an output interface deasserts the `ready` signal, the input interface receives the deasserted `ready` signal, as well. This functionality ensures that backpressure on the output interfaces is propagated to the input interface.



When the **Qualify Valid Out** option is enabled, the `out_valid` signals on all other output interfaces are gated when backpressure is applied from one output interface. In this case, when any output interface deasserts its `ready` signal, the `out_valid` signals on the other output interfaces are also deasserted.

When the **Qualify Valid Out** option is disabled, the output interfaces have a non-gated `out_valid` signal when backpressure is applied. In this case, when an output interface deasserts its `ready` signal, the `out_valid` signals on the other output interfaces are not affected.

Because the logic is combinational, the core introduces no latency.

## 14.5.2 Splitter Core Interfaces

The Avalon-ST Splitter core supports streaming data, with optional packet, channel, and error signals. The core propagates backpressure from any output interface to the input interface.

**Table 251. Avalon-ST Splitter Core Support**

Feature	Support
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

## 14.5.3 Splitter Core Parameters

**Table 252. Avalon-ST Splitter Core Parameters**

Parameter	Legal Values	Default Value	Description
<b>Number Of Outputs</b>	1 to 16	2	The number of output interfaces. Platform Designer supports 1 for some systems where no duplicated output is required.
<b>Qualify Valid Out</b>	Enabled, Disabled	Enabled	If enabled, the <code>out_valid</code> signal of all output interfaces is gated when back pressure is applied.
<b>Data Width</b>	1-512	8	The width of the data on the Avalon-ST data interfaces.
<b>Bits Per Symbol</b>	1-512	8	The number of bits per symbol for the input and output interfaces. For example, byte-oriented interfaces have 8-bit symbols.
<b>Use Packets</b>	Enabled, Disabled	Disabled	Enable support of data packet transfers. Packet support includes the <code>startofpacket</code> , <code>endofpacket</code> , and <code>empty</code> signals.
<b>Use Channel</b>	Enabled, Disabled	Disabled	Enable the channel signal.
<b>Channel Width</b>	0-8	1	The width of the <code>channel</code> signal on the data interfaces. This parameter is disabled when <b>Use Channel</b> is set to 0.

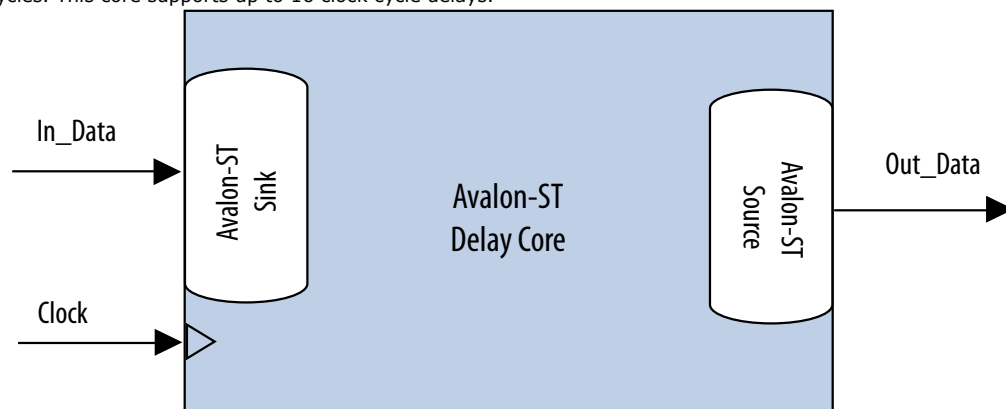
*continued...*

Parameter	Legal Values	Default Value	Description
Max Channels	0-255	1	The maximum number of channels that a data interface can support. This parameter is disabled when <b>Use Channel</b> is set to 0.
Use Error	Enabled, Disabled	Disabled	Enable the error signal.
Error Width	0-31	1	The width of the <code>error</code> signal on the output interfaces. A value of 0 indicates that the splitter core is not using the <code>error</code> signal. This parameter is disabled when <b>Use Error</b> is set to 0.

## 14.6 Avalon-ST Delay Core

**Figure 308. Avalon-ST Delay Core**

The Avalon-ST Delay Core provides a solution to delay Avalon-ST transactions by a constant number of clock cycles. This core supports up to 16 clock cycle delays.



The Avalon-ST Delay core adds a delay between the input and output interfaces. The core accepts transactions presented on the input interface and reproduces them on the output interface  $N$  cycles later without changing the transaction.

The input interface delays the input signals by a constant  $N$  number of clock cycles to the corresponding output signals of the output interface. The **Number Of Delay Clocks** parameter defines the constant  $N$ , which must be from 0 to 16. The change of the `in_valid` signal is reflected on the `out_valid` signal exactly  $N$  cycles later.

### 14.6.1 Delay Core Reset Signal

The Avalon-ST Delay core has a `reset` signal that is synchronous to the `clk` signal. When the core asserts the `reset` signal, the output signals are held at 0. After the `reset` signal is deasserted, the output signals are held at 0 for  $N$  clock cycles. The delayed values of the input signals are then reflected at the output signals after  $N$  clock cycles.

### 14.6.2 Delay Core Interfaces

The Delay core supports streaming data, with optional packet, channel, and error signals. The delay core does not support backpressure.





**Table 253. Avalon-ST Delay Core Support**

Feature	Support
Backpressure	Not supported.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

### 14.6.3 Delay Core Parameters

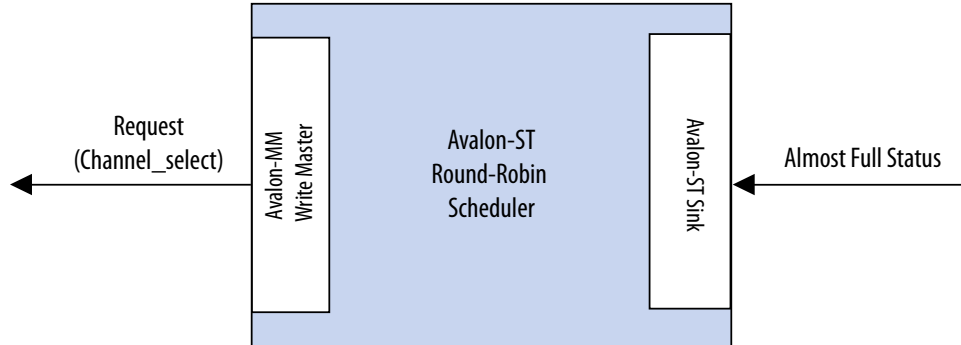
**Table 254. Avalon-ST Delay Core Parameters**

Parameter	Legal Values	Default Value	Description
<b>Number Of Delay Clocks</b>	0 to 16	1	Specifies the delay the core introduces, in clock cycles. Platform Designer supports 0 for some systems where no delay is required.
<b>Data Width</b>	1-512	8	The width of the data on the Avalon-ST data interfaces.
<b>Bits Per Symbol</b>	1-512	8	The number of bits per symbol for the input and output interfaces. For example, byte-oriented interfaces have 8-bit symbols.
<b>Use Packets</b>	0 or 1	0	Indicates whether data packet transfers are supported. Packet support includes the <code>startofpacket</code> , <code>endofpacket</code> , and <code>empty</code> signals.
<b>Use Channel</b>	0 or 1	0	The option to enable or disable the channel signal.
<b>Channel Width</b>	0-8	1	The width of the <code>channel</code> signal on the data interfaces. This parameter is disabled when <b>Use Channel</b> is set to 0.
<b>Max Channels</b>	0-255	1	The maximum number of channels that a data interface can support. This parameter is disabled when <b>Use Channel</b> is set to 0.
<b>Use Error</b>	0 or 1	0	The option to enable or disable the error signal.
<b>Error Width</b>	0-31	1	The width of the <code>error</code> signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when <b>Use Error</b> is set to 0.

## 14.7 Avalon-ST Round Robin Scheduler

**Figure 309. Avalon-ST Round Robin Scheduler**

The Avalon-ST Round Robin Scheduler core controls the read operations from a multi-channel Avalon-ST component that buffers data by channels. It reads the almost-full threshold values from the multiple channels in the multi-channel component and issues the read request to the Avalon-ST source according to a round-robin scheduling algorithm.



In a multi-channel component, the component can store data either in the sequence that it comes in (FIFO), or in segments according to the channel. When data is stored in segments according to channels, a scheduler is needed to schedule the read operations.

### 14.7.1 Almost-Full Status Interface (Round Robin Scheduler)

The Almost-Full Status interface is an Avalon-ST sink interface that collects the almost-full status from the sink components for the channels in the sequence provided.

**Table 255. Avalon-ST Interface Feature Support**

Feature	Property
Backpressure	Not supported
Data Width	Data width = 1; Bits per symbol = 1
Channel	Maximum channel = 32; Channel width = 5
Error	Not supported
Packet	Not supported

### 14.7.2 Request Interface (Round Robin Scheduler)

The Request Interface is an Avalon-MM write master interface that requests data from a specific channel. The Avalon-ST Round Robin Scheduler cycles through the channels it supports and schedules data to be read.

### 14.7.3 Round Robin Scheduler Operation

If a particular channel is almost full, the Avalon-ST Round Robin Scheduler does not schedule data to be read from that channel in the source component.



The scheduler only requests 1 bit of data from a channel at each transaction. To request 1 bit of data from channel  $n$ , the scheduler writes the value 1 to address ( $4 \times n$ ). For example, if the scheduler is requesting data from channel 3, the scheduler writes 1 to address  $0xC$ . At every clock cycle, the scheduler requests data from the next channel. Therefore, if the scheduler starts requesting from channel 1, at the next clock cycle, it requests from channel 2. The scheduler does not request data from a particular channel if the almost-full status for the channel is asserted. In this case, the scheduler uses one clock cycle without a request transaction.

The Avalon-ST Round Robin Scheduler cannot determine if the requested component is able to service the request transaction. The component asserts `waitrequest` when it cannot accept new requests.

**Table 256. Avalon-ST Round Robin Scheduler Ports**

Signal	Direction	Description
<b>Clock and Reset</b>		
<code>clk</code>	In	Clock reference.
<code>reset_n</code>	In	Asynchronous active low reset.
<b>Avalon-MM Request Interface</b>		
<code>request_address</code> ( $\log_2$ <code>Max_Channels-1:0</code> )	Out	The write address that indicates which channel has the request.
<code>request_write</code>	Out	Write enable signal.
<code>request_writedata</code>	Out	The amount of data requested from the particular channel. This value is always fixed at 1.
<code>request_waitrequest</code>	In	Wait request signal that pauses the scheduler when the slave cannot accept a new request.
<b>Avalon-ST Almost-Full Status Interface</b>		
<code>almost_full_valid</code>	In	Indicates that <code>almost_full_channel</code> and <code>almost_full_data</code> are valid.
<code>almost_full_channel</code> ( <code>Channel_Width-1:0</code> )	In	Indicates the channel for the current status indication.
<code>almost_full_data</code> ( $\log_2$ <code>Max_Channels-1:0</code> )	In	A 1-bit signal that is asserted high to indicate that the channel indicated by <code>almost_full_channel</code> is almost full.

## 14.7.4 Round Robin Scheduler Parameters

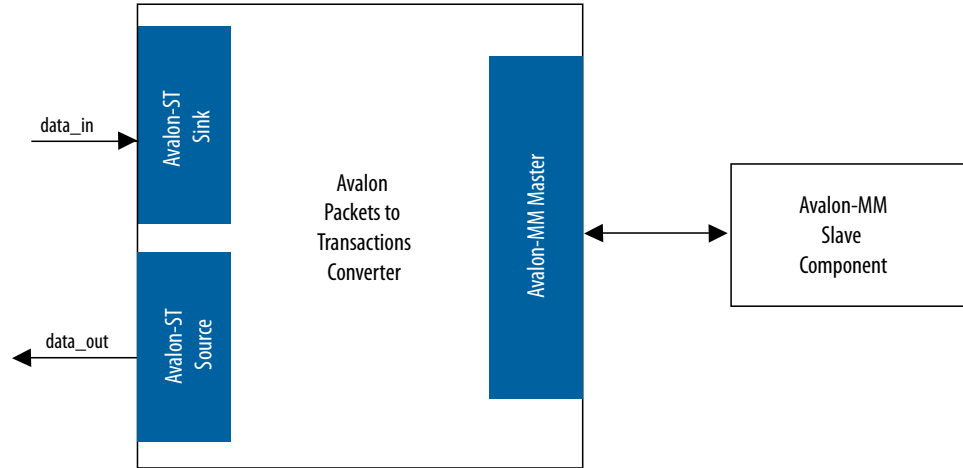
**Table 257. Avalon-ST Round Robin Scheduler Parameters**

Parameters	Legal Values	Default Value	Description
Number of channels	2-32	2	Specifies the number of channels the Avalon-ST Round Robin Scheduler supports.
Use almost-full status	Enabled, Disabled	Disabled	If enabled, the scheduler uses the almost-full interface. If not, the core requests data from the next channel at the next clock cycle.

## 14.8 Avalon Packets to Transactions Converter

**Figure 310. Avalon Packets to Transactions Converter Core**

The Avalon Packets to Transactions Converter core receives streaming data from upstream components and initiates Avalon-MM transactions. The core then returns Avalon-MM transaction responses to the requesting components.



**Note:** The SPI Slave to Avalon Master Bridge and JTAG to Avalon Master Bridge are examples of the Packets to Transactions Converter core. For more information, refer to the *Avalon Interface Specifications*.

### Related Links

[Avalon Interface Specifications](#)

### 14.8.1 Packets to Transactions Converter Interfaces

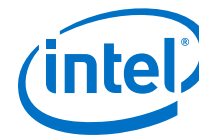
**Table 258. Properties of Avalon-ST Interfaces**

Feature	Property
Backpressure	Ready latency = 0.
Data Width	Data width = 8 bits; Bits per symbol = 8.
Channel	Not supported.
Error	Not used.
Packet	Supported.

The Avalon-MM master interface supports read and write transactions. The data width is set to 32 bits, and burst transactions are not supported.

### 14.8.2 Packets to Transactions Converter Operation

The Packets to Transactions Converter core receives streams of packets on its Avalon-ST sink interface and initiates Avalon-MM transactions. Upon receiving transaction responses from Avalon-MM slaves, the core transforms the responses to packets and returns them to the requesting components via its Avalon-ST source interface. The core does not report Avalon-ST errors.



### 14.8.2.1 Packets to Transactions Converter Data Packet Formats

A response packet is returned for every write transaction. The core also returns a response packet if a no transaction (0x7f) is received. An invalid transaction code is regarded as a no transaction. For read transactions, the core returns the data read.

The Packets to Transactions Converter core expects incoming data streams to be in the formats shown in the table below.

**Table 259. Data Packet Formats**

Byte	Field	Description
<b>Transaction Packet Format</b>		
0	Transaction code	Type of transaction.
1	Reserved	Reserved for future use.
[3:2]	Size	Transaction size in bytes. For write transactions, the size indicates the size of the data field. For read transactions, the size indicates the total number of bytes to read.
[7:4]	Address	32-bit address for the transaction.
[n:8]	Data	Transaction data; data to be written for write transactions.
<b>Response Packet Format</b>		
0	Transaction code	The transaction code with the most significant bit inverted.
1	Reserved	Reserved for future use.
[4:2]	Size	Total number of bytes read/written successfully.

#### Related Links

[Packets to Transactions Converter Interfaces](#) on page 972

### 14.8.2.2 Packets to Transactions Converter Supported Transactions

The Packets to Transactions Converter core supports the following Avalon-MM transactions:

**Table 260. Packets to Transactions Converter Supported Transactions**

Transaction Code	Avalon-MM Transaction	Description
0x00	Write, non-incrementing address.	Writes data to the address until the total number of bytes written to the same word address equals to the value specified in the size field.
0x04	Write, incrementing address.	Writes transaction data starting at the current address.
0x10	Read, non-incrementing address.	Reads 32 bits of data from the address until the total number of bytes read from the same address equals to the value specified in the size field.
0x14	Read, incrementing address.	Reads the number of bytes specified in the size parameter starting from the current address.
0x7f	No transaction.	No transaction is initiated. You can use this transaction type for testing purposes. Although no transaction is initiated on the Avalon-MM interface, the core still returns a response packet for this transaction code.

The Packets to Transactions Converter core can process only a single transaction at a time. The `ready` signal on the core's Avalon-ST sink interface is asserted only when the current transaction is completely processed.

No internal buffer is implemented on the datapaths. Data received on the Avalon-ST interface is forwarded directly to the Avalon-MM interface and vice-versa. Asserting the `waitrequest` signal on the Avalon-MM interface backpressures the Avalon-ST sink interface. In the opposite direction, if the Avalon-ST source interface is backpressured, the `read` signal on the Avalon-MM interface is not asserted until the backpressure is alleviated. Backpressuring the Avalon-ST source in the middle of a read can result in data loss. In this cases, the core returns the data that is successfully received.

A transaction is considered complete when the core receives an EOP. For write transactions, the actual data size is expected to be the same as the value of the `size` property. Whether or not both values agree, the core always uses the end of packet (EOP) to determine the end of data.

### 14.8.2.3 Packets to Transactions Converter Malformed Packets

The following are examples of malformed packets:

- **Consecutive start of packet (SOP)**—An SOP marks the beginning of a transaction. If an SOP is received in the middle of a transaction, the core drops the current transaction without returning a response packet for the transaction, and initiates a new transaction. This effectively precesses packets without an end of packet (EOP).
- **Unsupported transaction codes**—The core processes unsupported transactions as a no transaction.

## 14.9 Avalon-ST Streaming Pipeline Stage

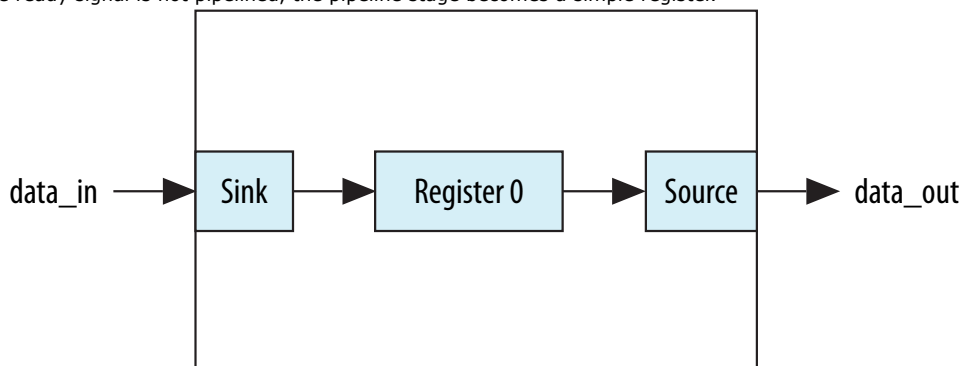
The Avalon-ST pipeline stage receives data from an Avalon-ST source interface, and outputs the data to an Avalon-ST sink interface. In the absence of back pressure, the Avalon-ST pipeline stage source interface outputs data one cycle after receiving the data on its sink interface.

If the pipeline stage receives back pressure on its source interface, it continues to assert its source interface's current data output. While the pipeline stage is receiving back pressure on its source interface and it receives new data on its sink interface, the pipeline stage internally buffers the new data. It then asserts back pressure on its sink interface.

After the backpressure is deasserted, the pipeline stage's source interface is deasserted and the pipeline stage asserts internally buffered data (if present). Additionally, the pipeline stage deasserts back pressure on its sink interface.

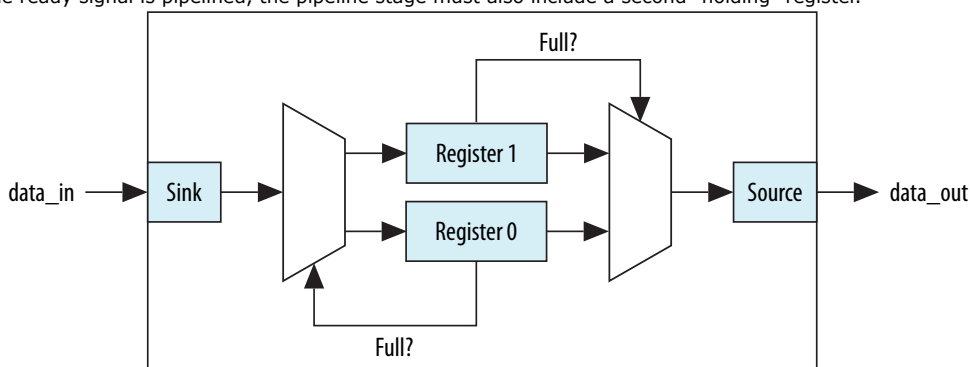
**Figure 311. Pipeline Stage Simple Register**

If the ready signal is not pipelined, the pipeline stage becomes a simple register.



**Figure 312. Pipeline Stage Holding Register**

If the ready signal is pipelined, the pipeline stage must also include a second "holding" register.



## 14.10 Streaming Channel Multiplexer and Demultiplexer Cores

The Avalon-ST channel multiplexer core receives data from various input interfaces and multiplexes the data into a single output interface, using the optional `channel` signal to indicate the origin of the data. The Avalon-ST channel demultiplexer core receives data from a channelized input interface and drives that data to multiple output interfaces, where the output interface is selected by the input `channel` signal.

The multiplexer and demultiplexer cores can transfer data between interfaces on cores that support unidirectional flow of data. The multiplexer and demultiplexer allow you to create multiplexed or demultiplexed datapaths without having to write custom HDL code. The multiplexer includes an Avalon-ST Round Robin Scheduler.

### Related Links

[Avalon-ST Round Robin Scheduler](#) on page 970

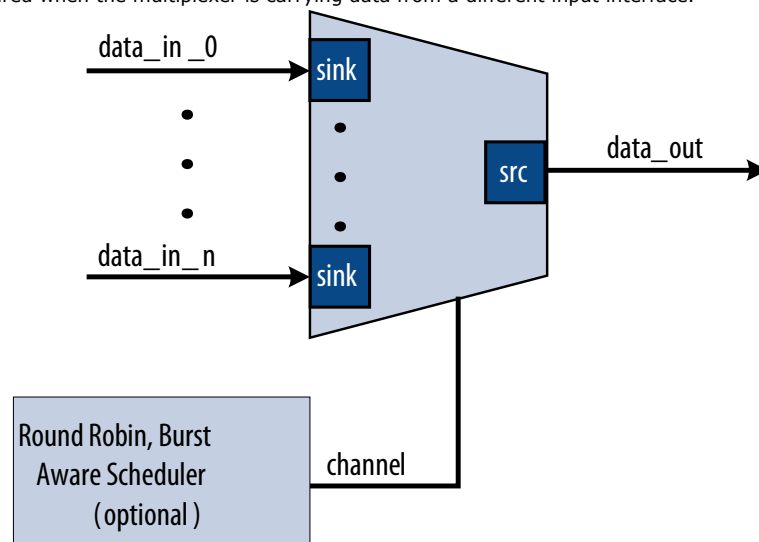
### 14.10.1 Software Programming Model For the Multiplexer and Demultiplexer Components

The multiplexer and demultiplexer components do not have any user-visible control or status registers. Therefore, Platform Designer cannot control or configure any aspect of the multiplexer or demultiplexer at run-time. The components cannot generate interrupts.

### 14.10.2 Avalon-ST Multiplexer

**Figure 313. Avalon-ST Multiplexer**

The Avalon-ST multiplexer takes data from a variety of input data interfaces, and multiplexes the data onto a single output interface. The multiplexer includes a round-robin scheduler that selects from the next input interface that has data. Each input interface has the same width as the output interface, so that the other input interfaces are backpressured when the multiplexer is carrying data from a different input interface.



The multiplexer includes an optional channel signal that enables each input interface to carry channelized data. The output interface channel width is equal to:

$$(\log_2 (n-1)) + 1 + w$$

where  $n$  is the number of input interfaces, and  $w$  is the channel width of each input interface. All input interfaces must have the same channel width. These bits are appended to either the most or least significant bits of the output channel signal.

The scheduler processes one input interface at a time, selecting it for transfer. Once an input interface has been selected, data from that input interface is sent until one of the following scenarios occurs:

- The specified number of cycles have elapsed.
- The input interface has no more data to send and the `valid` signal is deasserted on a ready cycle.
- When packets are supported, `endofpacket` is asserted.





### 14.10.2.1 Multiplexer Input Interfaces

Each input interface is an Avalon-ST data interface that optionally supports packets. The input interfaces are identical; they have the same symbol and data widths, error widths, and channel widths.

### 14.10.2.2 Multiplexer Output Interface

The output interface carries the multiplexed data stream with data from the inputs. The symbol, data, and error widths are the same as the input interfaces.

The width of the `channel` signal is the same as the input interfaces, with the addition of the bits needed to indicate the origin of the data.

You can configure the following parameters for the output interface:

- **Data Bits Per Symbol**—The bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—The number of symbols (words) that are transferred per beat (transfer). Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Channel Signal Width (bits)**— The number of bits Platform Designer uses for the channel signal for output interfaces. For example, set this parameter to 1 if you have two input interfaces with no channel, or set this parameter to 2 if you have two input interfaces with a channel width of 1 bit. The input channel can have a width between 0-31 bits.
- **Error Signal Width (bits)**—The width of the `error` signal for input and output interfaces. A value of 0 means the `error` signal is not in use.

*Note:* If you change only bits per symbol, and do not change the data width, errors are generated.

### 14.10.2.3 Multiplexer Parameters

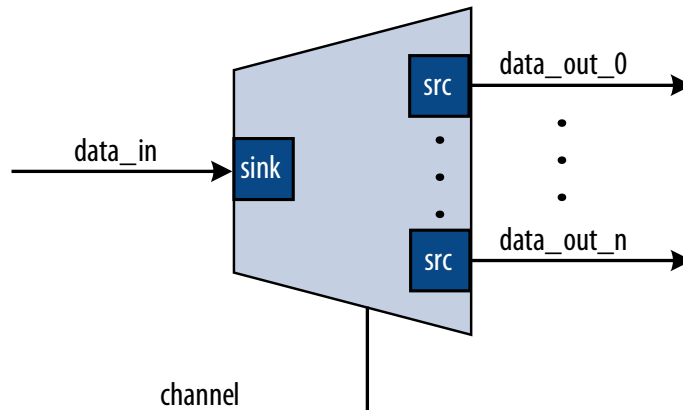
You can configure the following parameters for the multiplexer:

- **Number of Input Ports**—The number of input interfaces that the multiplexer supports. Valid values are 2 to 16.
- **Scheduling Size (Cycles)**—The number of cycles that are sent from a single channel before changing to the next channel.
- **Use Packet Scheduling**—When this parameter is turned on, the multiplexer only switches the selected input interface on packet boundaries. Therefore, packets on the output interface are not interleaved.
- **Use high bits to indicate source port**—When this parameter is turned on, the multiplexer uses the high bits of the output `channel` signal to indicate the origin of the input interface of the data. For example, if the input interfaces have 4-bit channel signals, and the multiplexer has 4 input interfaces, the output interface has a 6-bit channel signal. If this parameter is turned on, bits [5:4] of the output channel signal indicate origin of the input interface of the data, and bits [3:0] are the channel bits that were presented at the input interface.

### 14.10.3 Avalon-ST Demultiplexer

**Figure 314. Avalon-ST Demultiplexer**

That Avalon-ST demultiplexer takes data from a channelized input data interface and provides that data to multiple output interfaces, where the output interface selected for a particular transfer is specified by the input `channel` signal.



The data is delivered to the output interfaces in the same order it is received at the input interface, regardless of the value of `channel`, `packet`, `frame`, or any other signal. Each of the output interfaces has the same width as the input interface; each output interface is idle when the demultiplexer is driving data to a different output interface. The demultiplexer uses  $\log_2(\text{num\_output\_interfaces})$  bits of the `channel` signal to select the output for the data; the remainder of the channel bits are forwarded to the appropriate output interface unchanged.

#### 14.10.3.1 Demultiplexer Input Interface

Each input interface is an Avalon-ST data interface that optionally supports packets. You can configure the following parameters for the input interface:

- **Data Bits Per Symbol**—The bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—The number of symbols (words) that are transferred per beat (transfer). Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Channel Signal Width (bits)**—The number of bits for the `channel` signal for output interfaces. A value of 0 means that output interfaces do not use the optional `channel` signal.
- **Error Signal Width (bits)**—The width of the `error` signal for input and output interfaces. A value of 0 means the `error` signal is in use.

**Note:** If you change only bits per symbol, and do not change the data width, errors are generated.



### 14.10.3.2 Demultiplexer Output Interface

Each output interface carries data from a subset of channels from the input interface. Each output interface is identical; all have the same symbol and data widths, error widths, and channel widths. The symbol, data, and error widths are the same as the input interface. The width of the `channel` signal is the same as the input interface, without the bits that the demultiplexer uses to select the output interface.

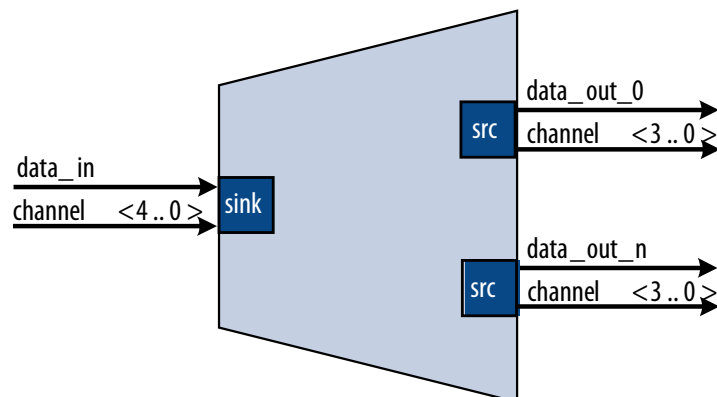
### 14.10.3.3 Demultiplexer Parameters

You can configure the following parameters for the demultiplexer:

- **Number of Output Ports**—The number of output interfaces that the multiplexer supports. Valid values are 2 to 16.
- **High channel bits select output**—When this option is turned on, the demultiplexing function uses the high bits of the input `channel` signal, and the low order bits are passed to the output. When this option is turned off, the demultiplexing function uses the low order bits, and the high order bits are passed to the output.

Where you place the signals in your design affects the functionality; for example, there is one input interface and two output interfaces. If the low-order bits of the channel signal select the output interfaces, the even channels go to channel 0, and the odd channels go to channel 1. If the high-order bits of the channel signal select the output interface, channels 0 to 7 go to channel 0 and channels 8 to 15 go to channel 1.

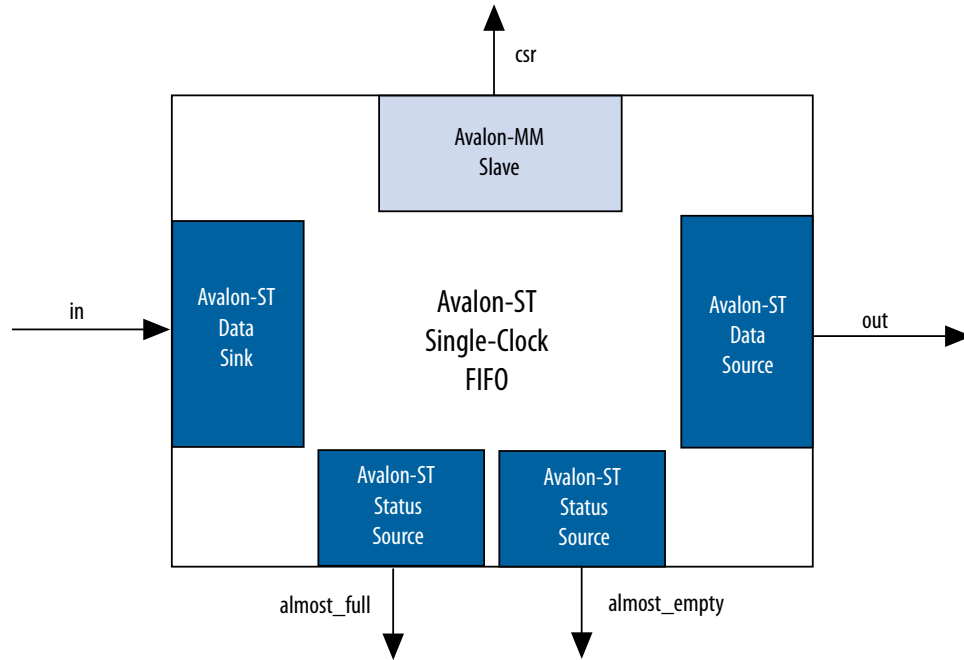
Figure 315. Select Bits for the Demultiplexer



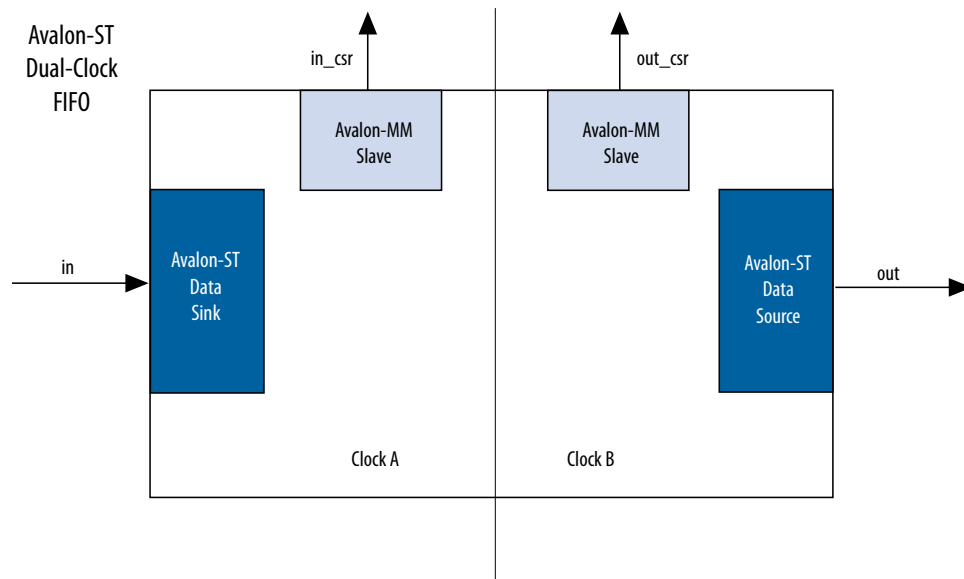
## 14.11 Single-Clock and Dual-Clock FIFO Cores

The Avalon-ST Single-Clock and Avalon-ST Dual-Clock FIFO cores are FIFO buffers which operate with a common clock and independent clocks for input and output ports respectively.

**Figure 316. Avalon-ST Single Clock FIFO Core**



**Figure 317. Avalon-ST Dual Clock FIFO Core**



### 14.11.1 Interfaces Implemented in FIFO Cores

The following interfaces are implemented in FIFO cores:

[Avalon-ST Data Interface](#) on page 981

[Avalon-MM Control and Status Register Interface](#) on page 981



Avalon-ST Status Interface on page 981

### 14.11.1.1 Avalon-ST Data Interface

Each FIFO core has an Avalon-ST data sink and source interfaces. The data sink and source interfaces in the dual-clock FIFO core are driven by different clocks.

**Table 261. Avalon-ST Interfaces Properties**

Feature	Property
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported, up to 255 channels.
Error	Configurable.
Packet	Configurable.

### 14.11.1.2 Avalon-MM Control and Status Register Interface

You can configure the single-clock FIFO core to include an optional Avalon-MM interface, and the dual-clock FIFO core to include an Avalon-MM interface in each clock domain. The Avalon-MM interface provides access to 32-bit registers, which allows you to retrieve the FIFO buffer fill level and configure the almost-empty and almost-full thresholds. In the single-clock FIFO core, you can also configure the packet and error handling modes.

### 14.11.1.3 Avalon-ST Status Interface

The single-clock FIFO core has two optional Avalon-ST status source interfaces from which you can obtain the FIFO buffer almost-full and almost empty statuses.

## 14.11.2 FIFO Operating Modes

- **Default mode**—The core accepts incoming data on the `in` interface (Avalon-ST data sink) and forwards it to the `out` interface (Avalon-ST data source). The core asserts the `valid` signal on the Avalon-ST source interface to indicate that data is available at the interface.
- **Store and forward mode**—This mode applies only to the single-clock FIFO core. The core asserts the `valid` signal on the `out` interface only when a full packet of data is available at the interface. In this mode, you can also enable the drop-on-error feature by setting the `drop_on_error` register to 1. When this feature is enabled, the core drops all packets received with the `in_error` signal asserted.
- **Cut-through mode**—This mode applies only to the single-clock FIFO core. The core asserts the `valid` signal on the `out` interface to indicate that data is available for consumption when the number of entries specified in the `cut_through_threshold` register are available in the FIFO buffer.

*Note:* To turn on **Cut-through mode**, the **Use store and forward** parameter must be set to 0. Turning on **Use store and forward mode** prompts the user to turn on **Use fill level**, and then the CSR appears.

### 14.11.3 Fill Level of the FIFO Buffer

You can obtain the fill level of the FIFO buffer via the optional Avalon-MM control and status interface. Turn on the **Use fill level** parameter (**Use sink fill level** and **Use source fill level** in the dual-clock FIFO core) and read the `fill_level` register.

The dual-clock FIFO core has two fill levels, one in each clock domain. Due to the latency of the clock crossing logic, the fill levels reported in the input and output clock domains may be different for any instance. In both cases, the fill level may report badly for the clock domain; that is, the fill level is reported high in the input clock domain, and low in the output clock domain.

The dual-clock FIFO has an output pipeline stage to improve  $f_{MAX}$ . This output stage is accounted for when calculating the output fill level, but not when calculating the input fill level. Therefore, the best measure of the amount of data in the FIFO is by the fill level in the output clock domain. The fill level in the input clock domain represents the amount of space available in the FIFO (available space = FIFO depth – input fill level).

### 14.11.4 Almost-Full and Almost-Empty Thresholds to Prevent Overflow and Underflow

You can use almost-full and almost-empty thresholds as a mechanism to prevent FIFO overflow and underflow. This feature is available only in the single-clock FIFO core. To use the thresholds, turn on the **Use fill level**, **Use almost-full status**, and **Use almost-empty status** parameters. You can access the `almost_full_threshold` and `almost_empty_threshold` registers via the csr interface and set the registers to an optimal value for your application.

You can obtain the almost-full and almost-empty statuses from `almost_full` and `almost_empty` interfaces (Avalon-ST status source). The core asserts the `almost_full` signal when the fill level is equal to or higher than the almost-full threshold. Likewise, the core asserts the `almost_empty` signal when the fill level is equal to or lower than the almost-empty threshold.

### 14.11.5 Single-Clock and Dual-Clock FIFO Core Parameters

**Table 262. Single-Clock and Dual-Clock FIFO Core Parameters**

Parameter	Legal Values	Description
<b>Bits per symbol</b>	1–32	These parameters determine the width of the FIFO. FIFO width = <b>Bits per symbol</b> * <b>Symbols per beat</b> , where: <b>Bits per symbol</b> is the number of bits in a symbol, and <b>Symbols per beat</b> is the number of symbols transferred in a beat.
<b>Symbols per beat</b>	1–32	
<b>Error width</b>	0–32	The width of the <code>error</code> signal.
<b>FIFO depth</b>	$2^n$	The FIFO depth. An output pipeline stage is added to the FIFO to increase performance, which increases the FIFO depth by one. <n> = n=1,2,3,4...
<b>Use packets</b>	—	Turn on this parameter to enable data packet support on the Avalon-ST data interfaces.
<b>Channel width</b>	1–32	The width of the <code>channel</code> signal.
<b>Avalon-ST Single Clock FIFO Only</b>		
<i>continued...</i>		



Parameter	Legal Values	Description
<b>Use fill level</b>	—	Turn on this parameter to include the Avalon-MM control and status register interface (CSR). The CSR is enabled when <b>Use fill level</b> is set to 1.
<b>Use Store and Forward</b>		To turn on <b>Cut-through mode</b> , <b>Use store and forward</b> must be set to 0. Turning on <b>Use store and forward</b> prompts the user to turn on <b>Use fill level</b> , and then the CSR appears.
<b>Avalon-ST Dual Clock FIFO Only</b>		
<b>Use sink fill level</b>	—	Turn on this parameter to include the Avalon-MM control and status register interface in the input clock domain.
<b>Use source fill level</b>	—	Turn on this parameter to include the Avalon-MM control and status register interface in the output clock domain.
<b>Write pointer synchronizer length</b>	2–8	The length of the write pointer synchronizer chain. Setting this parameter to a higher value leads to better metastability while increasing the latency of the core.
<b>Read pointer synchronizer length</b>	2–8	The length of the read pointer synchronizer chain. Setting this parameter to a higher value leads to better metastability.
<b>Use Max Channel</b>	—	Turn on this parameter to specify the maximum channel number.
<b>Max Channel</b>	1–255	Maximum channel number.

**Note:** For more information about metastability in Intel devices, refer to *Understanding Metastability in FPGAs*. For more information about metastability analysis and synchronization register chains, refer to the *Managing Metastability*.

#### Related Links

- [Managing Metastability with the Intel Quartus Prime Software](#) on page 986
- [Understanding Metastability in FPGAs](#)

### 14.11.6 Avalon-ST Single-Clock FIFO Registers

**Table 263. Avalon-ST Single-Clock FIFO Registers**

The CSR interface in the Avalon-ST Single Clock FIFO core provides access to registers.

32-Bit Word Offset	Name	Access	Reset	Description
0	fill_level	R	0	24-bit FIFO fill level. Bits 24 to 31 are not used.
1	Reserved	—	—	Reserved for future use.
2	almost_full_threshold	RW	<b>FIFO depth</b> –1	Set this register to a value that indicates the FIFO buffer is getting full.
3	almost_empty_threshold	RW	0	Set this register to a value that indicates the FIFO buffer is getting empty.
4	cut_through_threshold	RW	0	<b>0</b> —Enables store and forward mode.

*continued...*

32-Bit Word Offset	Name	Access	Reset	Description
				<p><b>Greater than 0</b>—Enables cut-through mode and specifies the minimum of entries in the FIFO buffer before the <code>valid</code> signal on the Avalon-ST source interface is asserted. Once the FIFO core starts sending the data to the downstream component, it continues to do so until the end of the packet.</p> <p><i>Note:</i> To turn on <b>Cut-through mode</b>, <b>Use store and forward mode</b> must be set to 0. Turning on <b>Use store and forward mode</b> prompts the user to turn on <b>Use fill level</b>, and then the CSR appears.</p>
5	drop_on_error	RW	0	<p><b>0</b>—Disables drop-on error.  <b>1</b>—Enables drop-on error.</p> <p>This register applies only when the <b>Use packet</b> and <b>Use store and forward</b> parameters are turned on.</p>

**Table 264. Register Description for Avalon-ST Dual-Clock FIFO**

The `in_csr` and `out_csr` interfaces in the Avalon-ST Dual Clock FIFO core reports the FIFO fill level.

32-Bit Word Offset	Name	Access	Reset Value	Description
0	fill_level	R	0	24-bit FIFO fill level. Bits 24 to 31 are not used.

**Related Links**

- [Avalon Memory-Mapped Design Optimizations](#)
- [Avalon Interface Specifications](#)

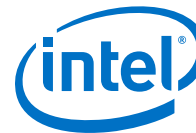
## 14.12 Document Revision History

**Table 265. Document Revision History**

The table below indicates edits made to the *Platform Designer System Design Components* content since its creation.

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>• Changed instances of <i>Qsys Pro</i> to <i>Platform Designer</i>.</li> <li>• Changed instances of <i>AXI Default Slave</i> to <i>Error Response Slave</i>.</li> <li>• Updated topics: Error Response Slave.</li> <li>• Updated Figure: Error Response Slave Parameter Editor.</li> <li>• Added Figure: Error Response Slave Parameter Editor with Enabled CSR Support.</li> <li>• Updated topics: CSR Registers and renamed to Error Response Slave CSR Registers.</li> <li>• Added topic: Error Response Slave Access Violation Service.</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>• Implemented Intel rebranding.</li> <li>• Implemented Qsys rebranding.</li> </ul>
2016.05.03	16.0.0	<p>Updated Address Span Extender</p> <ul style="list-style-type: none"> <li>• Address Span Extender register mapping better explained</li> <li>• Address Span Extender Parameters table added</li> <li>• Address Span Extender example added</li> </ul>
2015.11.02	15.1.0	Changed instances of <i>Quartus II</i> to <i>Quartus Prime</i> .
2015.05.04	15.0.0	Avalon-MM Unaligned Burst Expansion Bridge and Avalon-MM Pipeline Bridge, <b>Maximum pending read transactions</b> parameter. Extended description.
<i>continued...</i>		





Date	Version	Changes
December 2014	14.1.0	<ul style="list-style-type: none"> <li>• AXI Timeout Bridge.</li> <li>• Added notes to <i>Avalon-MM Clock Crossing Bridge</i> pertaining to:               <ul style="list-style-type: none"> <li>— SDC constraints for its internal asynchronous FIFOs.</li> <li>— FIFO-based clock crossing.</li> </ul> </li> </ul>
June 2014	14.0.0	<ul style="list-style-type: none"> <li>• AXI Bridge support.</li> <li>• Address Span Extender updates.</li> <li>• Avalon-MM Unaligned Burst Expansion Bridge support.</li> </ul>
November 2013	13.1.0	<ul style="list-style-type: none"> <li>• Address Span Extender</li> </ul>
May 2013	13.0.0	<ul style="list-style-type: none"> <li>• Added Streaming Pipeline Stage support.</li> <li>• Added AMBA APB support.</li> </ul>
November 2012	12.1.0	<ul style="list-style-type: none"> <li>• Moved relevant content from the <i>Embedded Peripherals IP User Guide</i>.</li> </ul>

### Related Links

#### Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 15 Managing Metastability with the Intel Quartus Prime Software

---

You can use the Intel Quartus Prime software to analyze the average mean time between failures (MTBF) due to metastability caused by synchronization of asynchronous signals, and optimize the design to improve the metastability MTBF.

All registers in digital devices, such as FPGAs, have defined signal-timing requirements that allow each register to correctly capture data at its input ports and produce an output signal. To ensure reliable operation, the input to a register must be stable for a minimum amount of time before the clock edge (register setup time or  $t_{SU}$ ) and a minimum amount of time after the clock edge (register hold time or  $t_H$ ). The register output is available after a specified clock-to-output delay ( $t_{CO}$ ).

If the data violates the setup or hold time requirements, the output of the register might go into a metastable state. In a metastable state, the voltage at the register output hovers at a value between the high and low states, which means the output transition to a defined high or low state is delayed beyond the specified  $t_{CO}$ . Different destination registers might capture different values for the metastable signal, which can cause the system to fail.

In synchronous systems, the input signals must always meet the register timing requirements, so that metastability does not occur. Metastability problems commonly occur when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the signal can arrive at any time relative to the destination clock.

The MTBF due to metastability is an estimate of the average time between instances when metastability could cause a design failure. A high MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design. You should determine an acceptable target MTBF in the context of your entire system and taking in account that MTBF calculations are statistical estimates.

The metastability MTBF for a specific signal transfer, or all the transfers in a design, can be calculated using information about the design and the device characteristics. Improving the metastability MTBF for your design reduces the chance that signal transfers could cause metastability problems in your device.

The Intel Quartus Prime software provides analysis, optimization, and reporting features to help manage metastability in Intel designs. These metastability features are supported only for designs constrained with the Intel Quartus Prime Timing Analyzer. Both typical and worst-case MBTF values are generated for select device families.

### Related Links

- [Understanding Metastability in FPGAs](#)  
For more information about metastability due to signal synchronization, its effects in FPGAs, and how MTBF is calculated



- [Reliability Report](#)  
For information about Intel device reliability

## 15.1 Metastability Analysis in the Intel Quartus Prime Software

When a signal transfers between circuitry in unrelated or asynchronous clock domains, the first register in the new clock domain acts as a synchronization register.

To minimize the failures due to metastability in asynchronous signal transfers, circuit designers typically use a sequence of registers (a synchronization register chain or synchronizer) in the destination clock domain to resynchronize the signal to the new clock domain and allow additional time for a potentially metastable signal to resolve to a known value. Designers commonly use two registers to synchronize a new signal, but a standard of three registers provides better metastability protection.

The timing analyzer can analyze and report the MTBF for each identified synchronizer that meets its timing requirements, and can generate an estimate of the overall design MTBF. The software uses this information to optimize the design MTBF, and you can use this information to determine whether your design requires longer synchronizer chains.

### Related Links

- [Metastability and MTBF Reporting](#) on page 989
- [MTBF Optimization](#) on page 992

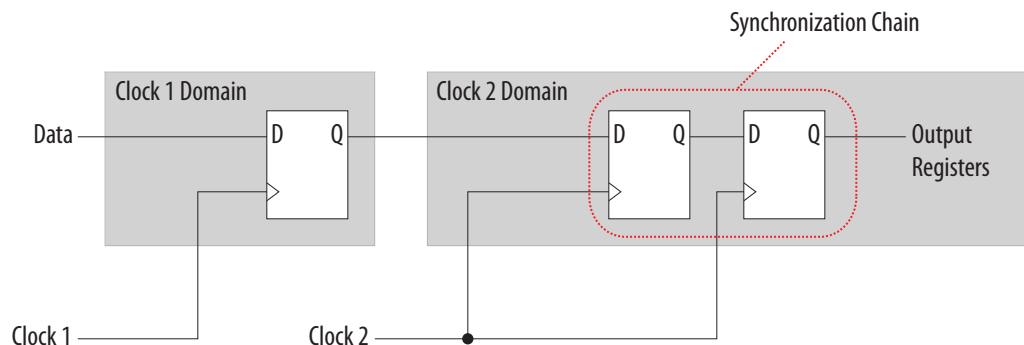
### 15.1.1 Synchronization Register Chains

A synchronization register chain, or synchronizer, is defined as a sequence of registers that meets the following requirements:

- The registers in the chain are all clocked by the same clock or phase-related clocks.
- The first register in the chain is driven asynchronously or from an unrelated clock domain.
- Each register fans out to only one register, except the last register in the chain.

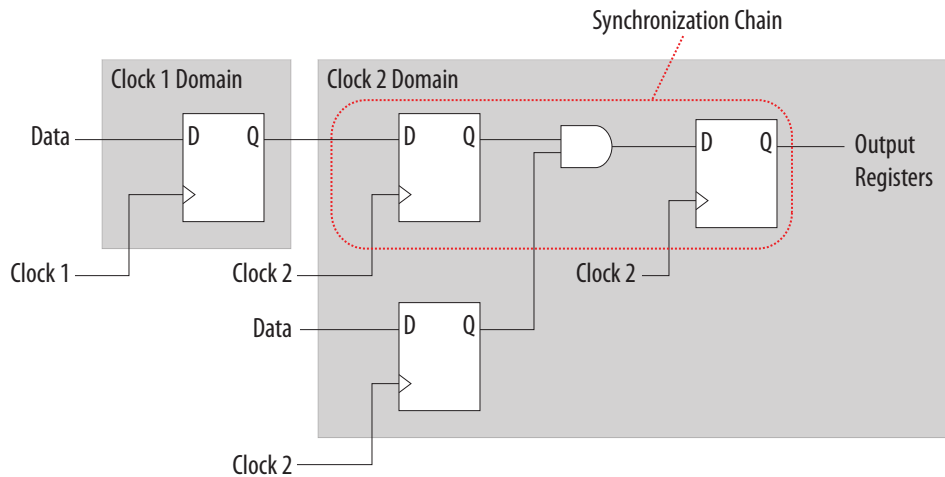
The length of the synchronization register chain is the number of registers in the synchronizing clock domain that meet the above requirements. The figure shows a sample two-register synchronization chain.

**Figure 318. Sample Synchronization Register Chain**



The path between synchronization registers can contain combinational logic if all registers of the synchronization register chain are in the same clock domain. The figure shows an example of a synchronization register chain that includes logic between the registers.

**Figure 319. Sample Synchronization Register Chain Containing Logic**



The timing slack available in the register-to-register paths of the synchronizer allows a metastable signal to settle, and is referred to as the available settling time. The available settling time in the MTBF calculation for a synchronizer is the sum of the output timing slacks for each register in the chain. Adding available settling time with additional synchronization registers improves the metastability MTBF.

**Related Links**

[How Timing Constraints Affect Synchronizer Identification and Metastability Analysis](#) on page 988

**15.1.2 Identify Synchronizers for Metastability Analysis**

The first step in enabling metastability MTBF analysis and optimization in the Intel Quartus Prime software is to identify which registers are part of a synchronization register chain. You can apply synchronizer identification settings globally to automatically list possible synchronizers with the **Synchronizer identification** option on the **Timing Analyzer** page in the **Settings** dialog box.

Synchronization chains are already identified within most Intel FPGA intellectual property (IP) cores.

**Related Links**

[Identify Synchronizers for Metastability Analysis](#) on page 988

**15.1.3 How Timing Constraints Affect Synchronizer Identification and Metastability Analysis**

The timing analyzer can analyze metastability MTBF only if a synchronization chain meets its timing requirements. The metastability failure rate depends on the timing slack available in the synchronizer’s register-to-register connections, because that



slack is the available settling time for a potential metastable signal. Therefore, you must ensure that your design is correctly constrained with the real application frequency requirements to get an accurate MTBF report.

In addition, the **Auto** and **Forced If Asynchronous** synchronizer identification options use timing constraints to automatically detect the synchronizer chains in the design. These options check for signal transfers between circuitry in unrelated or asynchronous clock domains, so clock domains must be related correctly with timing constraints.

The timing analyzer views input ports as asynchronous signals unless they are associated correctly with a clock domain. If an input port fans out to registers that are not acting as synchronization registers, apply a `set_input_delay` constraint to the input port; otherwise, the input register might be reported as a synchronization register. Constraining a synchronous input port with a `set_max_delay` constraint for a setup ( $t_{SU}$ ) requirement does not prevent synchronizer identification because the constraint does not associate the input port with a clock domain.

Instead, use the following command to specify an input setup requirement associated with a clock:

```
set_input_delay -max -clock <clock name> <latch - launch - tsu  
requirement> <input port name>
```

Registers that are at the end of false paths are also considered synchronization registers because false paths are not timing-analyzed. Because there are no timing requirements for these paths, the signal may change at any point, which may violate the  $t_{SU}$  and  $t_H$  of the register. Therefore, these registers are identified as synchronization registers. If these registers are not used for synchronization, you can turn off synchronizer identification and analysis. To do so, set **Synchronizer Identification** to **Off** for the first synchronization register in these register chains.

## 15.2 Metastability and MTBF Reporting

The Intel Quartus Prime software reports the metastability analysis results in the Compilation Report and Timing Analyzer reports.

The MTBF calculation uses timing and structural information about the design, silicon characteristics, and operating conditions, along with the data toggle rate.

If you change the **Synchronizer Identification** settings, you can generate new metastability reports by rerunning the timing analyzer. However, you should rerun the Fitter first so that the registers identified with the new setting can be optimized for metastability MTBF.

### Related Links

- [Metastability Reports](#) on page 990
- [MTBF Optimization](#) on page 992
- [Synchronizer Data Toggle Rate in MTBF Calculation](#) on page 992
- [Understanding Metastability in FPGAs](#)  
For more information about how metastability MTBF is calculated



## 15.2.1 Metastability Reports

Metastability reports provide summaries of the metastability analysis results. In addition to the MTBF Summary and Synchronizer Summary reports, the Timing Analyzer tool reports additional statistics in a report for each synchronizer chain.

*Note:* If the design uses only the **Auto Synchronizer Identification** setting, the reports list likely synchronizers but do not report MTBF. To obtain an MTBF for each register chain, force identification of synchronization registers.

*Note:* If the synchronizer chain does not meet its timing requirements, the reports list identified synchronizers but do not report MTBF. To obtain MTBF calculations, ensure that the design is properly constrained and that the synchronizer meets its timing requirements.

### Related Links

- [Identify Synchronizers for Metastability Analysis](#) on page 988
- [How Timing Constraints Affect Synchronizer Identification and Metastability Analysis](#) on page 988

### 15.2.1.1 MTBF Summary Report

The MTBF Summary reports an estimate of the overall robustness of cross-clock domain and asynchronous transfers in the design. This estimate uses the MTBF results of all synchronization chains in the design to calculate an MTBF for the entire design.

#### 15.2.1.1.1 Typical and Worst-Case MTBF of Design

The MTBF Summary Report shows the **Typical MTBF of Design** and the **Worst-Case MTBF of Design** for supported fully-characterized devices. The typical MTBF result assumes typical conditions, defined as nominal silicon characteristics for the selected device speed grade, as well as nominal operating conditions. The worst case MTBF result uses the worst case silicon characteristics for the selected device speed grade.

When you analyze multiple timing corners in the timing analyzer, the MTBF calculation may vary because of changes in the operating conditions, and the timing slack or available metastability settling time. Intel recommends running multi-corner timing analysis to ensure that you analyze the worst MTBF results, because the worst timing corner for MTBF does not necessarily match the worst corner for timing performance.

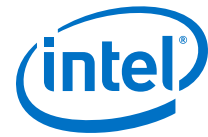
### Related Links

[Timing Analyzer page](#)

#### 15.2.1.1.2 Synchronizer Chains

The MTBF Summary report also lists the **Number of Synchronizer Chains Found** and the length of the **Shortest Synchronizer Chain**, which can help you identify whether the report is based on accurate information.

If the number of synchronizer chains found is different from what you expect, or if the length of the shortest synchronizer chain is less than you expect, you might have to add or change **Synchronizer Identification** settings for the design. The report also provides the **Worst Case Available Settling Time**, defined as the available settling time for the synchronizer with the worst MTBF.



You can use the reported **Fraction of Chains for which MTBFs Could Not be Calculated** to determine whether a high proportion of chains are missing in the metastability analysis. A fraction of 1, for example, means that MTBF could not be calculated for any chains in the design. MTBF is not calculated if you have not identified the chain with the appropriate **Synchronizer identification** option, or if paths are not timing-analyzed and therefore have no valid slack for metastability analysis. You might have to correct your timing constraints to enable complete analysis of the applicable register chains.

### 15.2.1.1.3 Increasing Available Settling Time

The MTBF Summary report specifies how an increase of 100ps in available settling time increases the MTBF values. If your MTBF is not satisfactory, this metric can help you determine how much extra slack would be required in your synchronizer chain to allow you to reach the desired design MTBF.

### 15.2.1.2 Synchronizer Summary Report

The **Synchronizer Summary** lists the synchronization register chains detected in the design depending on the Synchronizer Identification setting.

The **Source Node** is the register or input port that is the source of the asynchronous transfer. The **Synchronization Node** is the first register of the synchronization chain. The **Source Clock** is the clock domain of the source node, and the **Synchronization Clock** is the clock domain of the synchronizer chain.

This summary reports the calculated **Worst-Case MTBF**, if available, and the **Typical MTBF**, for each appropriately identified synchronization register chain that meets its timing requirement.

#### Related Links

[Synchronizer Chain Statistics Report in the Timing Analyzer](#) on page 991

### 15.2.1.3 Synchronizer Chain Statistics Report in the Timing Analyzer

The timing analyzer provides an additional report for each synchronizer chain.

The **Chain Summary** tab matches the Synchronizer Summary information described in the Synchronizer Summary Report, while the **Statistics** tab adds more details. These details include whether the **Method of Synchronizer Identification** was **User Specified** (with the **Forced if Asynchronous** or **Forced** settings for the **Synchronizer Identification** setting), or **Automatic** (with the **Auto** setting). The **Number of Synchronization Registers in Chain** report provides information about the parameters that affect the MTBF calculation, including the **Available Settling Time** for the chain and the **Data Toggle Rate Used in MTBF Calculation**.

The following information is also included to help you locate the chain in your design:

- **Source Clock** and **Asynchronous Source** node of the signal.
- **Synchronization Clock** in the destination clock domain.
- Node names of the **Synchronization Registers** in the chain.

#### Related Links

[Synchronizer Data Toggle Rate in MTBF Calculation](#) on page 992

## 15.2.2 Synchronizer Data Toggle Rate in MTBF Calculation

The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency. That is, the arriving data is assumed to switch once every eight source clock cycles.

If multiple clocks apply, the highest frequency is used. If no source clocks can be determined, the data rate is taken as 12.5% of the synchronization clock frequency.

If you know an approximate rate at which the data changes, specify it with the **Synchronizer Toggle Rate** assignment in the Assignment Editor. You can also apply this assignment to an entity or the entire design. Set the data toggle rate, in number of transitions per second, on the first register of a synchronization chain. The timing analyzer takes the specified rate into account when computing the MTBF of that particular register chain. If a data signal never toggles and does not affect the reliability of the design, you can set the **Synchronizer Toggle Rate** to **0** for the synchronization chain so the MTBF is not reported. To apply the assignment with Tcl, use the following command:

```
set_instance_assignment -name SYNCHRONIZER_TOGGLE_RATE <toggle rate in transitions/second> -to <register name>
```

In addition to **Synchronizer Toggle Rate**, there are two other assignments associated with toggle rates, which are not used for metastability MTBF calculations. The I/O Maximum Toggle Rate is only used for pins, and specifies the worst-case toggle rates used for signal integrity purposes. The Power Toggle Rate assignment is used to specify the expected time-averaged toggle rate, and is used by the Power Analyzer to estimate time-averaged power consumption.

## 15.3 MTBF Optimization

In addition to reporting synchronization register chains and MTBF values found in the design, the Intel Quartus Prime software can also protect these registers from optimizations that might negatively impact MTBF and can optimize the register placement and routing if the MTBF is too low.

Synchronization register chains must first be explicitly identified as synchronizers. Intel recommends that you set **Synchronizer Identification** to **Forced If Asynchronous** for all registers that are part of a synchronizer chain.

Optimization algorithms, such as register duplication and logic retiming in physical synthesis, are not performed on identified synchronization registers. The Fitter protects the number of synchronization registers specified by the **Synchronizer Register Chain Length** option.

In addition, the Fitter optimizes identified synchronizers for improved MTBF by placing and routing the registers to increase their output setup slack values. Adding slack in the synchronizer chain increases the available settling time for a potentially metastable signal, which improves the chance that the signal resolves to a known value, and exponentially increases the design MTBF. The Fitter optimizes the number of synchronization registers specified by the **Synchronizer Register Chain Length** option.





Metastability optimization is **on** by default. To view or change the **Optimize Design for Metastability** option, click **Assignments > Settings > Compiler Settings > Advanced Settings (Fitter)**. To turn the optimization on or off with Tcl, use the following command:

```
set_global_assignment -name OPTIMIZE_FOR_METASTABILITY <ON|OFF>
```

### Related Links

[Identify Synchronizers for Metastability Analysis](#) on page 988

## 15.3.1 Synchronization Register Chain Length

The **Synchronization Register Chain Length** option specifies how many registers should be protected from optimizations that might reduce MTBF for each register chain, and controls how many registers should be optimized to increase MTBF with the **Optimize Design for Metastability** option.

For example, if the **Synchronization Register Chain Length** option is set to **2**, optimizations such as register duplication or logic retiming are prevented from being performed on the first two registers in all identified synchronization chains. The first two registers are also optimized to improve MTBF when the **Optimize Design for Metastability** option is turned on.

The default setting for the **Synchronization Register Chain Length** option is **3**. The first register of a synchronization chain is always protected from operations that might reduce MTBF, but you should set the protection length to protect more of the synchronizer chain. Intel recommends that you set this option to the maximum length of synchronization chains you have in your design so that all synchronization registers are preserved and optimized for MTBF.

Click **Assignments > Settings > Compiler Settings > Advanced Settings (Synthesis)** to change the global **Synchronization Register Chain Length** option.

You can also set the **Synchronization Register Chain Length** on a node or an entity in the Assignment Editor. You can set this value on the first register in a synchronization chain to specify how many registers to protect and optimize in this chain. This individual setting is useful if you want to protect and optimize extra registers that you have created in a specific synchronization chain that has low MTBF, or optimize less registers for MTBF in a specific chain where the maximum frequency or timing performance is not being met. To make the global setting with Tcl, use the following command:

```
set_global_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers>
```

To apply the assignment to a design instance or the first register in a specific chain with Tcl, use the following command:

```
set_instance_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers> -to <register or instance name>
```



## 15.4 Reducing Metastability Effects

You can check your design's metastability MTBF in the Metastability Summary report, and determine an acceptable target MTBF given the context of your entire system and the fact that MTBF calculations are statistical estimates. A high metastability MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design.

This section provides guidelines to ensure complete and accurate metastability analysis, and some suggestions to follow if the Intel Quartus Prime metastability reports calculate an unacceptable MTBF value. The Timing Optimization Advisor (available from the Tools menu) gives similar suggestions in the Metastability Optimization section.

### Related Links

[Metastability Reports](#) on page 990

### 15.4.1 Apply Complete System-Centric Timing Constraints for the Timing Analyzer

To enable the Intel Quartus Prime metastability features, make sure that the timing analyzer is used for timing analysis.

Ensure that the design is fully timing constrained and that it meets its timing requirements. If the synchronization chain does not meet its timing requirements, MTBF cannot be calculated. If the clock domain constraints are set up incorrectly, the signal transfers between circuitry in unrelated or asynchronous clock domains might be identified incorrectly.

Use industry-standard system-centric I/O timing constraints instead of using FPGA-centric timing constraints.

You should use `set_input_delay` constraints in place of `set_max_delay` constraints to associate each input port with a clock domain to help eliminate false positives during synchronization register identification.

### Related Links

[How Timing Constraints Affect Synchronizer Identification and Metastability Analysis](#) on page 988

### 15.4.2 Force the Identification of Synchronization Registers

Use the guidelines in "*Identifying Synchronizers for Metastability Analysis*" to ensure the software reports and optimizes the appropriate register chains.

Identify synchronization registers with the **Synchronizer Identification** set to **Forced If Asynchronous** in the Assignment Editor. If there are any registers that the software detects as synchronous but you want to be analyzed for metastability, apply the **Forced** setting to the first synchronizing register. Set **Synchronizer Identification** to **Off** for registers that are not synchronizers for asynchronous signals or unrelated clock domains.



To help you find the synchronizers in your design, you can set the global **Synchronizer Identification** setting on the **Timing Analyzer** page of the **Settings** dialog box to **Auto** to generate a list of all the possible synchronization chains in your design.

#### Related Links

[Identify Synchronizers for Metastability Analysis](#) on page 988

### 15.4.3 Set the Synchronizer Data Toggle Rate

The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency.

To obtain a more accurate MTBF for a specific chain or all chains in your design, set the **Synchronizer Toggle Rate**.

#### Related Links

[Synchronizer Data Toggle Rate in MTBF Calculation](#) on page 992

### 15.4.4 Optimize Metastability During Fitting

Ensure that the **Optimize Design for Metastability** setting is turned on.

#### Related Links

[MTBF Optimization](#) on page 992

### 15.4.5 Increase the Length of Synchronizers to Protect and Optimize

Increase the Synchronizer Chain Length parameter to the maximum length of synchronization chains in your design. If you have synchronization chains longer than 2 identified in your design, you can protect the entire synchronization chain from operations that might reduce MTBF and allow metastability optimizations to improve the MTBF.

#### Related Links

[Synchronization Register Chain Length](#) on page 993

### 15.4.6 Increase the Number of Stages Used in Synchronizers

Designers commonly use two registers in a synchronization chain to minimize the occurrence of metastable events, and a standard of three registers provides better metastability protection. However, synchronization chains with two or even three registers may not be enough to produce a high enough MTBF when the design runs at high clock and data frequencies.

If a synchronization chain is reported to have a low MTBF, consider adding an additional register stage to your synchronization chain. This additional stage increases the settling time of the synchronization chain, allowing more opportunity for the signal to resolve to a known state during a metastable event. Additional settling time increases the MTBF of the chain and improves the robustness of your design. However, adding a synchronization stage introduces an additional stage of latency on the signal.



If you use the Altera FIFO IP core with separate read and write clocks to cross clock domains, increase the metastability protection (and latency) for better MTBF. In the DCFIFO parameter editor, choose the **Best metastability protection, best fmax, unsynchronized clocks** option to add three or more synchronization stages. You can increase the number of stages to more than three using the **How many sync stages?** setting.

### 15.4.7 Select a Faster Speed Grade Device

The design MTBF depends on process parameters of the device used. Faster devices are less susceptible to metastability issues. If the design MTBF falls significantly below the target MTBF, switching to a faster speed grade can improve the MTBF substantially.

## 15.5 Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Intel Quartus Prime Command-Line and Tcl API Help browser.

To run the Help browser, type the following command at the command prompt and then press Enter:

```
quartus_sh --qhelp
```

### Related Links

- [Tcl Scripting](#)  
For more information about Tcl scripting
- [Intel Quartus Prime Settings File Reference Manual](#)  
For more information about settings and constraints in the Intel Quartus Prime software
- [Command-Line Scripting](#)  
For more information about command-line scripting
- [About Intel Quartus Prime Scripting](#)  
For more information about command-line scripting

### 15.5.1 Identifying Synchronizers for Metastability Analysis

To apply the global Synchronizer Identification assignment, use the following command:

```
set_global_assignment -name SYNCHRONIZER_IDENTIFICATION <OFF|AUTO|"FORCED IF ASYNCHRONOUS">
```

To apply the **Synchronizer Identification** assignment to a specific register or instance, use the following command:

```
set_instance_assignment -name SYNCHRONIZER_IDENTIFICATION <AUTO|"FORCED IF ASYNCHRONOUS"|FORCED|OFF> -to <register or instance name>
```



## 15.5.2 Synchronizer Data Toggle Rate in MTBF Calculation

To specify a toggle rate for MTBF calculations as described on page “[R\\*\\*Synchronizer Data Toggle Rate in MTBF Calculation](#)”, use the following command:

```
set_instance_assignment -name SYNCHRONIZER_TOGGLE_RATE <toggle rate in transitions/second> -to <register name>
```

### Related Links

[Synchronizer Data Toggle Rate in MTBF Calculation](#) on page 992

## 15.5.3 report\_metastability and Tcl Command

If you use a command-line or scripting flow, you can generate the metastability analysis reports described in “[C\\*\\*Metastability Reports](#)” outside of the Intel Quartus Prime and user interfaces.

The table describes the options for the `report_metastability` and Tcl command.

**Table 266. report\_metastability Command Options**

Option	Description
-append	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten.
-file <name>	Sends the results to an ASCII or HTML file. The extension specified in the file name determines the file type — either <b>*.txt</b> or <b>*.html</b> .
-panel_name <name>	Sends the results to the panel and specifies the name of the new panel.
-stdout	Indicates the report be sent to the standard output, via messages. This option is required only if you have selected another output format, such as a file, and would also like to receive messages.

### Related Links

[Metastability Reports](#) on page 990

## 15.5.4 MTBF Optimization

To ensure that metastability optimization described on page “[C\\*\\*MTBF Optimization](#)” is turned on (or to turn it off), use the following command:

```
set_global_assignment -name OPTIMIZE_FOR_METASTABILITY <ON|OFF>
```

### Related Links

[MTBF Optimization](#) on page 992



### 15.5.5 Synchronization Register Chain Length

To globally set the number of registers in a synchronization chain to be protected and optimized as described on page “[C\\*\\*Synchronization Register Chain Length](#)”, use the following command:

```
set_global_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers>
```

To apply the assignment to a design instance or the first register in a specific chain, use the following command:

```
set_instance_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers> -to <register or instance name>
```

#### Related Links

[Synchronization Register Chain Length](#) on page 993

## 15.6 Managing Metastability

Intel’s Intel Quartus Prime software provides industry-leading analysis and optimization features to help you manage metastability in your FPGA designs. Set up your Intel Quartus Prime project with the appropriate constraints and settings to enable the software to analyze, report, and optimize the design MTBF. Take advantage of these features in the Intel Quartus Prime software to make your design more robust with respect to metastability.

## 15.7 Document Revision History

**Table 267. Document Revision History**

Date	Version	Changes
2017.11.06	17.1.0	<ul style="list-style-type: none"><li>Corrected broken links to other documents.</li></ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"><li>Implemented Intel rebranding.</li></ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"><li>Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li></ul>
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
June 2014	14.0.0	Updated formatting.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template.
July 2010	10.0.0	Technical edit.
November 2009	9.1.0	Clarified description of synchronizer identification settings. Minor changes to text and figures throughout document.
March 2009	9.0.0	Initial release.



### **Related Links**

#### [Documentation Archive](#)

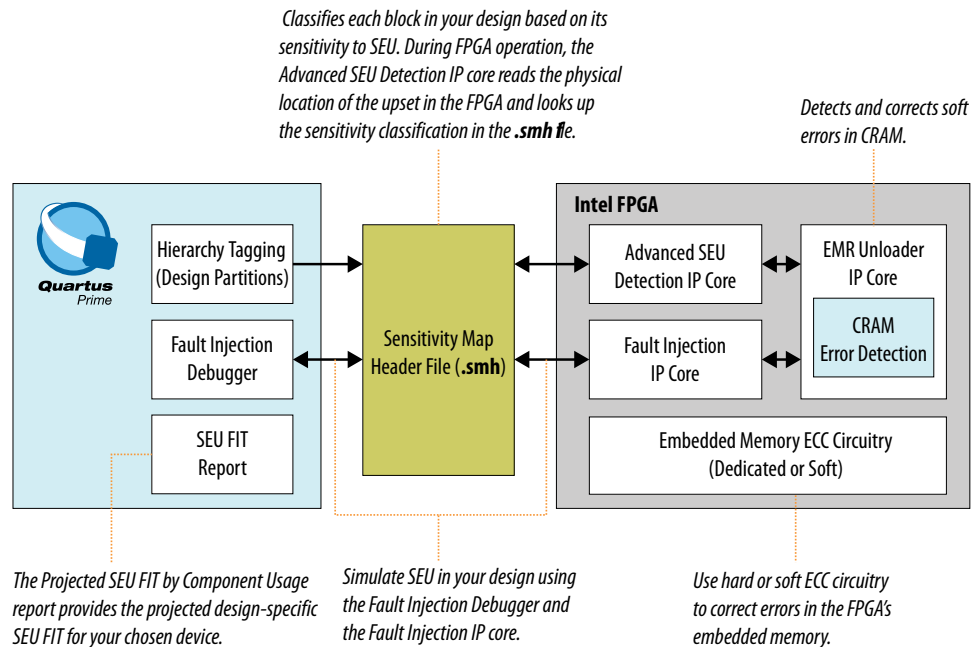
For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

## 16 Mitigating Single Event Upset

Single event upsets (SEUs) are rare, unintended changes in the state of an FPGA's internal memory elements caused by cosmic radiation effects. The change in state is a soft error and the FPGA incurs no permanent damage. Because of the unintended memory state, the FPGA may operate erroneously until background scrubbing fixes the upset.

The Intel Quartus Prime software offers several features to detect and correct the effects of SEU, or soft errors, as well as to characterize the effects of SEU on your designs. Additionally, some Intel FPGAs contain dedicated circuitry to help detect and correct errors.

**Figure 320. Tools, IP, and Circuitry for Detecting and Correcting SEU**



Intel FPGAs have memory in user logic (block memory and registers) and in Configuration Random Access Memory (CRAM). The Intel Quartus Prime Programmer loads the CRAM with a .sof file. Then, the CRAM configures all FPGA logic and routing. If an SEU strikes a CRAM bit, the effect can be harmless if the device does not use the CRAM bit. However, the effect can be severe if the SEU affects critical logic or internal signal routing.

Often, a design does not require SEU mitigation because of the low chance of occurrence. However, for highly complex systems, such as systems with multiple high-density components, the error rate may be a significant system design factor. If your





system includes multiple FPGAs and requires very high reliability and availability, you should consider the implications of soft errors. Use the techniques in this chapter to detect and recover from these types of errors.

#### Related Links

- [Introduction to Single Event Upsets](#)
- [Understanding Single Event Functional Interrupts in FPGA Designs White Paper](#)

## 16.1 Failure Rates

The Soft Error Rate (SER) or SEU reliability is expressed in Failure in Time (FIT) units. One FIT unit is one soft error occurrence per billion hours of operation.

- For example, a design with 5,000 FIT experiences a mean of 5,000 SEU events in 1 billion hours (or 8,333.33 years). Because SEU events are statistically independent, FIT is additive: if a single FPGA has 5,000 FIT, then 10 FPGAs have 50,000 FIT (or 50K failures in 8,333 years).

Another reliability measurement is the mean time to failure (MTTF), which is the reciprocal of the FIT or 1/FIT.

- For a FIT of 5,000 in standard units of failures/billion hours, MTTF is:  
 $1 / (5,000/1\text{Bh}) = 1 \text{ billion} / 5,000 = 200,000 \text{ hours} = 22.83 \text{ years}$

SEU events follow a Poisson distribution, and the cumulative distribution function (CDF) for mean time between failures (MTBF) is an exponential distribution. For more information about failure rate calculation, refer to the *Intel FPGA Reliability Report*.

Neutron SEU incidence varies by altitude, latitude, and other environmental factors. The Intel Quartus Prime software provides SEU FIT reports based on compiles for sea level in Manhattan, New York. The JESD 89A specification defines the test parameters.

#### Tip:

You can convert the data to other locations and altitudes using calculators, such as those at [www.seutest.com](http://www.seutest.com). Additionally, you can adjust the SEU rates in your project by including the relative neutron flux (calculated at [www.seutest.com](http://www.seutest.com)) in your project's .qsf file.

#### Related Links

- [SEU FIT Parameters Report](#) on page 1010
- [JEDEC Standard 89A](#)
- <http://seutest.com/>  
Soft-error Testing Resources and Calculator
- [Intel FPGA Reliability Report](#)

## 16.2 Mitigating SEU Effects in Embedded User RAM

In the Intel Quartus Prime Pro Edition software, the implementation of the error correcting code (ECC) depends on the memory block where the RAM IP is instantiated. A RAM IP that is instantiated in a M20K block uses dedicated ECC circuitry. Conversely, a RAM IP that is instantiated in a LUTRAM uses a soft IP to implement the ECC. This is valid for all devices that the Intel Quartus Prime Pro Edition software supports.

You can reduce the FIT rate for these memories to near zero by enabling the ECC encode/decode blocks. On ingress, the ECC encoder adds 8 bits of redundancy to a 32 bit word. On egress, the decoder converts the 40 bit word back to 32 bits. You use the redundant bits to detect and correct errors in the data resulting from SEU.

The existence of hard ECC and the strength of the ECC code (number of corrected and detected bits) varies by device family. Refer to the device handbook for details. If a device does not have a hard ECC block you can add ECC parity or use an ECC IP core.

The SRAM memories associated with processor subsystems, such as for SoC devices, contain dedicated hard ECC. You do not need to take action to protect these memories.

For more information about embedded memories and ECC, refer to the *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*. For Intel Stratix 10 devices, refer to the *Intel Stratix 10 Embedded Memory User Guide*.

### Related Links

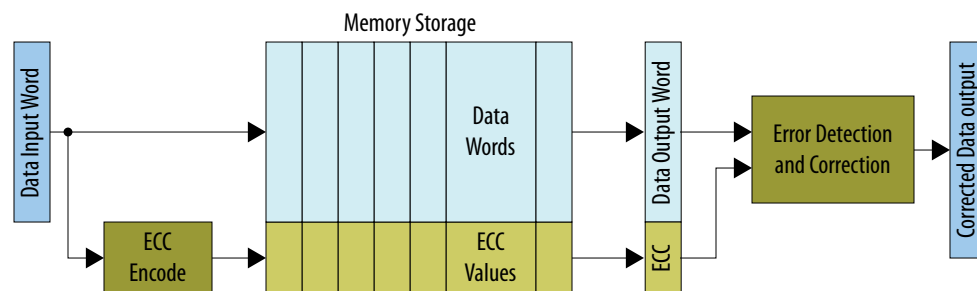
- [Error Correction Code](#)  
In *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*
- [Memory Blocks Error Correction Code Support](#)  
In *Intel Stratix 10 Embedded Memory User Guide*

## 16.2.1 Configuring RAM to Enable ECC

To enable ECC, configure the RAM as a 2-port RAM with independent read and write addresses. Using this feature does not reduce the available logic.

Although the ECC checking function results in some additional output delay, the hard ECC has a much higher  $f_{MAX}$  compared with an equivalent soft ECC block implemented in general logic. Additionally, you can pipeline the hard IP in the M20K block by configuring the ECC-enabled RAM to use an output register at the corrected data output port. This implementation increases performance and adds latency. For devices without dedicated circuitry, you can implement ECC by instantiating the ALTECC IP core, which performs ECC generation and checking functions.

Figure 321. Memory Storage and ECC



### Related Links

[ALTECC \(Error Correction Code: Encoder/Decoder\)](#)

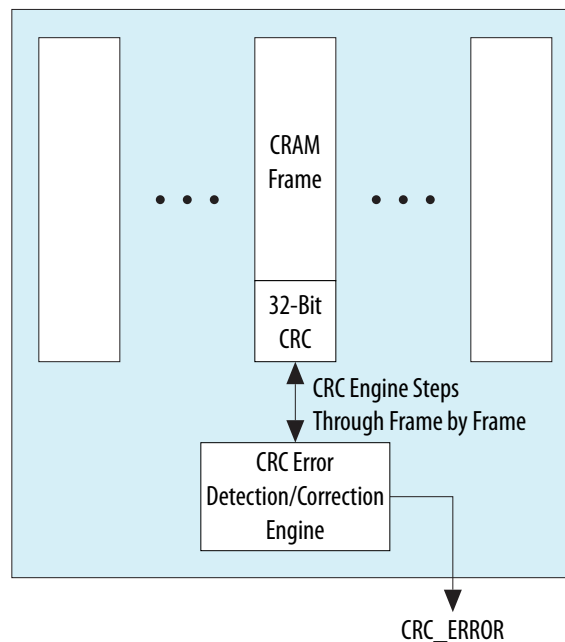
## 16.3 Mitigating SEU Effects in Configuration RAM (Intel Arria 10 and Intel Cyclone 10 GX devices)

Intel Arria 10 and Intel Cyclone 10 GX devices contain error detect CRC (EDCRC) hard blocks. These blocks detect and correct soft errors in CRAM, and are similar to those that protect internal user memory.

Intel FPGAs contain frames of CRAM. The size and number of frames is device specific. The device continually checks the CRAM frames for errors by loading each frame into a data register. The EDCRC block checks the frame for errors.

When the FPGA finds a soft error, the FPGA asserts its `CRC_ERROR` pin. You can monitor this pin in your system. When your system detects that the FPGA asserted this pin during operation, indicating the FPGA detected a soft error in the configuration RAM, the system can take action to recover from the error. For example, the system can perform a soft reset (after waiting for background scrubbing), reprogram the FPGA, or classify the error as benign and ignore it.

Figure 322. CRAM Frame



To enable error detection, point to **Assignments > Device > Device and Pin Options > Error Detection CRC**, and turn on error detection settings.

### Related Links

- [Single Level Upsets](#)
- [CRAM Error Detection Settings Reference](#) on page 1015

## 16.4 Mitigating SEU Effects in Configuration RAM (Intel Stratix 10 devices)

The Intel Stratix 10 device overlays the rows and columns with sectors to address the core logic for configuration and security. Each sector has its own EDC circuitry that calculates and uses parity syndrome for sector error detection and correction. The Local Sector Manager (LSM) controls and manages the EDC circuitry, and communicates with the Secure Device Manager (SDM) through the configuration network.

To enable error detection, point to **Assignments > Device > Device and Pin Options > Error Detection CRC**, and turn on error detection settings.

### 16.4.1 Error Message Register

Intel Stratix 10 devices store a maximum of four different error messages in queue when detecting an SEU error. You can retrieve these error messages through the Error Message Register (EMR). The EMR contains information on the error count in the queue, the sector address, error type, and the location of the error. You can shift out the contents of the EMR using:

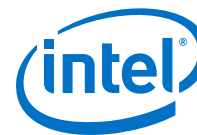
- Fault Injection Debugger tool
- Intel Stratix 10 Advanced SEU Detection IP core

**Table 268. Error Message Register Description**

Name	Width	Bit	Description						
Error count	32	—	Number of SEU error queue before processing READ_SEU_ERROR command (0 if no SEU errors are on the queue)						
Sector address	32	31:24	Reserved						
		23:16	The sector number of the sector with the error						
		15:4	Reserved						
		3:0	The number of errors detected in the sector						
Error location <sup>(13)</sup>	32	31:29	Error type: <table border="1" data-bbox="831 1356 1036 1486"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Single bit</td> </tr> <tr> <td>2</td> <td>Multi-bit</td> </tr> </tbody> </table>	Value	Description	1	Single bit	2	Multi-bit
		Value	Description						
1	Single bit								
2	Multi-bit								
		28	Correction Status: <table border="1" data-bbox="831 1570 1052 1701"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Not corrected</td> </tr> <tr> <td>1</td> <td>Corrected</td> </tr> </tbody> </table>	Value	Description	0	Not corrected	1	Corrected
Value	Description								
0	Not corrected								
1	Corrected								

*continued...*

<sup>(13)</sup> The error location provides the bit position for single bit errors only. For multiple bit errors, bit [23:0] returns 0.



Name	Width	Bit	Description
		27:24	Reserved
		23:12	Bit position within frame
		11:0	Combined of Row and Frame index (0-2336)

The SDM stores the SEU error information in an error queue that behaves like a stack, so the last error injected is the first to be dequeued.

### 16.4.2 SEU\_ERROR Pin Behavior

The SEU\_ERROR signal goes high whenever the error message queue contains one or more error messages. The signal stays high if there is an error message in the queue. The SEU\_ERROR signal goes low only when the SEU error message queue is empty which happens after you shift out all the error messages.

You must set to the SEU\_ERROR pin function to observe the SEU\_ERROR pin behavior.

## 16.5 Internal Scrubbing

In the Intel Quartus Prime Pro Edition software, all FPGA families support automatic CRAM error correction without reloading the original CRAM contents from an external copy of the original .sof.

For Intel Arria 10 and Intel Cyclone 10 GX devices, the internal scrubbing feature corrects single-bit and double-adjacent errors automatically. For Intel Stratix 10 devices, the internal scrubbing feature corrects only single-bit errors.

For Intel Arria 10 and Intel Cyclone 10 GX devices, if the FPGA finds a CRC error in a CRAM frame, the FPGA reconstructs the frame from the error correcting code calculated for that frame. Then the FPGA writes the correct frame into the CRAM.

Intel Stratix 10 devices calculate parity during configuration, and use this information to reconstruct the correct bit value.

*Note:* When the system detects a correctable upset, the correction is automatic. However, internal scrubbing cannot fix the device to a known good state.

If you enable internal scrubbing, you must still plan a recovery sequence. Although scrubbing can restore the CRAM array to intended configuration, latency occurs between the soft error detection and correction. During this latency period, the FPGA may operate with errors. If the FPGA must scan a large number of configuration bits, this latency can be up to 100 milliseconds. For more information about latency refer to the device datasheet.

To enable internal scrubbing, click **Assignments > Device > Device and Pin Options > Error Detection CRC** and turn on the **Enable internal scrubbing** option.

#### Related Links

- [Error Detection CRC Page](#)  
In *Intel Quartus Prime Help*
- [SEU Recovery](#) on page 1006



- [CRAM Error Detection Settings Reference](#) on page 1015

## 16.6 SEU Recovery

After correcting a CRAM bit flip, the FPGA is in its original configuration with respect to logic and routing. However, the FPGA may have an illegal internal state, for example, if the SEU error affects the function or operation of the circuit, resulting in erroneous output.

Errors due to faulty operation can propagate elsewhere within the FPGA or to the system outside the FPGA. During your design process, determine the possible SEU outcomes and design a recovery response that considers resetting the FPGA to a known state.

For more information, refer to *Recovering from a Single Event Upset* section of the *Intel Stratix 10 SEU Mitigation User Guide*.

### Related Links

[Recovering from a Single Event Upset](#)  
In *Intel Stratix 10 SEU Mitigation User Guide*

### 16.6.1 Planning for SEU Recovery

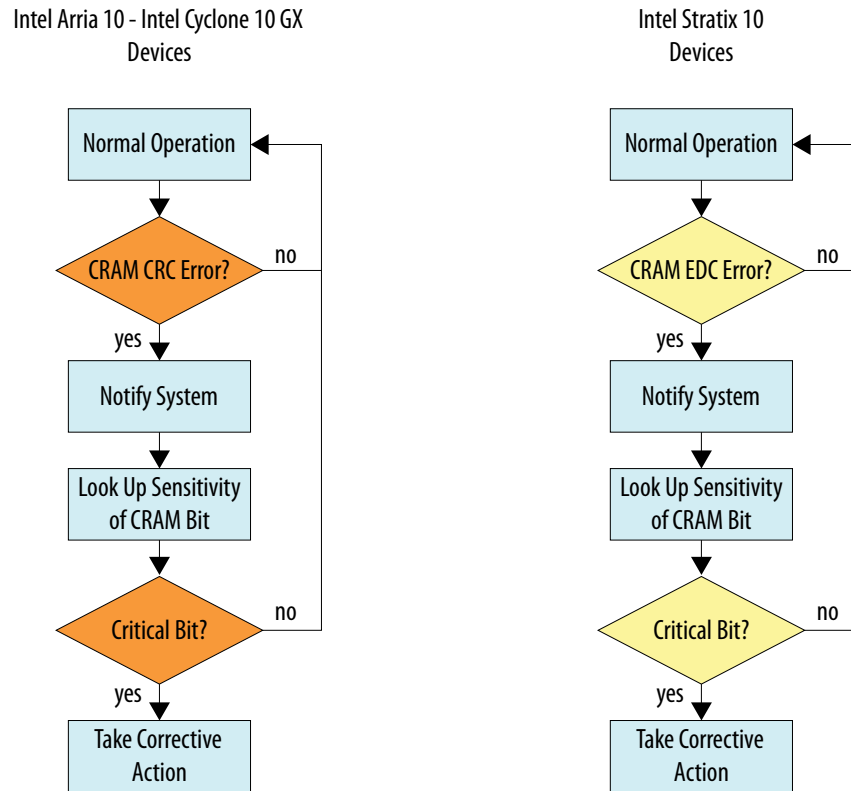
Reconfiguring a running FPGA typically has a significant system impact. When planning for SEU recovery, you must account for the time required to bring the FPGA to a state consistent with the current state of the system. For example, an internal state machine that is in an illegal state may require reset. Also, the surrounding logic may need to account for this unexpected operation.

Often, an SEU impacts CRAM bits that the implemented design does not use (for example, CRAM bits that control unused logic and routing wires). Depending on the implementation, FPGAs with high utilization only use about 40% of available CRAM bits. Therefore, only 40% of potential SEU events in the entire FPGA require intervention, and you can ignore the remaining 60%. Designs that do not completely fill the FPGA use even fewer available CRAM bits.

You can determine which portions of the implemented design are not critical to the FPGA's function. Examples include test circuitry that is not important to the FPGA operation, or other non-critical functions that the system can log but does not need to reprogram or reset.



Figure 323. Sensitivity Processing Flow



#### Related Links

- [AN 737: SEU Detection and Recovery in Intel Arria 10 Devices](#)
- [SEU Sensitivity Processing](#)  
In *Intel Stratix 10 SEU Mitigation User Guide*
- [Understanding Single Event Functional Interrupts in FPGA Designs White Paper](#)

### 16.6.2 Designating the Sensitivity of the Design Hierarchy

In the Intel Quartus Prime software, you indicate the criticality of each logic block by generating partitions, and assigning a sensitivity ID tag to each partition. The Intel Quartus Prime software stores this information in a Sensitivity Map Header File (.smh).

When an error occurs during system operation, the system determines the impact of the error by looking up the classification in the .smh file. The system can then take corrective action based on the classification.

**Note:** You must have a licensed version of Intel Quartus Prime software to generate .smh files.

To access the .smh file, you must add an instance of the Advanced SEU Detection IP core to your design.

### Related Links

- [Advanced SEU Detection IP Core User Guide](#)
- [Intel Stratix 10 SEU Mitigation User Guide](#)

#### 16.6.2.1 Hierarchy Tagging

The Intel Quartus Prime hierarchy tagging feature allows you to improve design-effective FIT rate by tagging only the critical logic for device operation.

You can also define the system recovery procedure based on knowledge of logic impaired by SEU. This technique reduces downtime for the FPGA and the system in which the FPGA resides. Other advantages of hierarchy tagging are:

- Increases system stability by avoiding disruptive recovery procedures for inconsequential errors.
- Allows diverse corrective action for different design logic.

The `.smh` file contains a mask for design sensitive bits in a compressed format. The Intel Quartus Prime software generates the sensitivity mask for the entire design.

### Related Links

[SEU Mitigation in Intel FPGA Devices: Hierarchy Tagging Online Course](#)

#### 16.6.2.2 Using Partitions to Specify Logic Sensitivity ID

1. In the Intel Quartus Prime software, designate a design block as a design partition.
2. Specify the sensitivity ID assigned to the partition in the **ASD Region** column in the **Design Partitions** window.

**Figure 324. ASD Region Column in the Design Partitions Window**

Partition Name	Hierarchy Path	Type	Preservation Level	Empty	Color	ASD Region
<<new>>						
state_m	inst1	Reconfigurable	Not Set	No	Red	0
taps	inst	Default	synthesized	No	Yellow	3
hvalues	inst2	Periphery Reuse Core	final	No	Green	2

Assign the partition a numeric sensitivity value from 0 to 16. The value represents the sensitivity tag associated with the partition.

- A sensitivity tag of 1 is the same as no assignment, and indicates a basic sensitivity level, which is "region used in design". If a soft error occurs in this partition, the Intel FPGA Advanced SEU Detection IP core reports the error as a critical error in the sensitivity region 1.
- A sensitivity tag of 0 is reserved, and indicates unused CRAM bits. You can explicitly set a partition to 0 to indicate that the partition is not critical. This setting excludes the partition from sensitivity mapping.

**Note:** You can use the same sensitivity tag for multiple design partitions.





Alternatively, use the following assignment:

```
set_global_assignment -name PARTITION_ASD_REGION_ID <asd_id> -section_id  
<partition_name>
```

### 16.6.3 Advanced SEU Detection IP Core

To correct and detect SEU in the FPGA CRAM, you must instantiate the Advanced SEU Detection IP core. When the FPGA's EDC detects an SEU, the Advanced SEU Detection IP core looks up the sensitivity of the affected bit in the `.smh` file.

- During system operation, the Advanced SEU Detection IP core reads the FPGA's error message register (EMR) to determine the location of the error.
- The IP core finds the upset location in the `.smh` file.
- The IP core returns whether or not the bit is critical for the design.

You can implement either an on-chip or external sensitivity processor:

- *On-chip sensitivity processor:* the IP core looks up the bit sensitivity in the `.smh` with a user-supplied memory interface.
- *External sensitivity processor:* the IP core notifies external logic (typically via a system CPU interrupt request), and provides cached event message register values to the off-chip sensitivity processor. The external sensitivity processor's memory system stores the `.smh` information.

The *Advanced SEU Detection IP Core User Guide* provides instructions for incorporating the IP core into your design, and describes how to access the `.smh` file.

#### Related Links

- [Advanced SEU Detection IP Core User Guide](#)
- [Intel Stratix 10 SEU Mitigation User Guide](#)

#### 16.6.3.1 On-Chip Sensitivity Processor

When you implement an on-chip sensitivity processor, the Advanced SEU Detection IP core interacts with user-supplied external memory access logic to read the `.smh` stored in external memory. Once it determines the sensitivity of the affected CRAM bit, the IP core can assert a critical error signal so that the system provides an appropriate response. If the SEU is not critical, the critical error signal may be left unasserted.

On-chip sensitivity processing is autonomous: the FPGA determines whether an SEU affected it without using external logic. On-chip sensitivity processing requires some FPGA logic resources for the external memory interface.

#### 16.6.3.2 External Sensitivity Processor

When you implement an external sensitivity processor, a CPU (such as the ARM processor in Intel SoC devices) receives an interrupt request when the FPGA detects an SEU. The CPU then reads the FPGA's error message register and looks up the bit sensitivity in the `.smh` stored in the CPU's memory space.



With external sensitivity processing, the FPGA does not need to implement an external memory interface or store the .smh. If the system already has a CPU, external sensitivity processing may be more hardware efficient than on-chip processing.

## 16.7 Intel Quartus Prime Software SEU FIT Reports

The Intel Quartus Prime software generates reports that contain the parameters involved in SEU FIT calculations and the result of these calculations for each component. These reports are available only for licensed users.

*Note:* These reports are not available for Intel Stratix 10 devices.

### 16.7.1 SEU FIT Parameters Report

The SEU FIT Parameters report shows the environmental assumptions that influence the FIT/Mb values.

Figure 325. SEU FIT Parameters

SEU FIT Parameters		
	Parameter	value
1	Device	5SGXEA7N2F45I2
2	Altitude	0.00
3	Neutron Flux	JESD - 89A assuming sea - level(> 10 MeV) 13.00 n / hr / cm2
4	Neutron Flux Multiplier	1.000
5	Alpha Flux	0.001 CPH/cm^2

*Change the Neutron Flux Multiplier using the assignment:  
set\_global\_assignment RELATIVE\_NEUTRON\_FLUX <relative\_flux>*

- **Altitude** represents the default altitude (above sea-level).
- **Neutron Flux Multiplier** is the relative flux for the default location, which is New York City per JESD specification. The default is 1. Change the setting by adding the following assignment to your .qsf file:

```
set_global_assignment RELATIVE_NEUTRON_FLUX <relative_flux>
```

*Note:* You can compute scaled values using the JESD published equations for altitude, latitude, and longitude. Websites, such as [www.seutest.com](http://www.seutest.com), can make this computation for you.

- **Alpha Flux** is the default for standard Intel packages; you cannot override the default.

*Note:* When you change the relative **Neutron Flux Multiplier**, the Intel Quartus Prime software only scales the neutron component of FIT. Location does not affect the Alpha flux.

#### Related Links

<http://seutest.com/>

Soft-error Testing Resources and Calculator



## 16.7.2 Projected SEU FIT by Component Usage Report

The Projected SEU FIT by Component Usage report shows the different components (or cell types) that comprise the total FIT rate, and SEU FIT calculation results.

An Intel FPGA's sensitivity to soft errors varies by process technology, component type, and your design choices when implementing the component (such as tradeoffs between area/delay and SEU rates). The report shows all bits (the raw FIT), utilized bits (only resources the design actually uses), and the ECC-mitigated bits.

**Figure 326. Projected SEU FIT by Component Usage Report**

Projected SEU FIT by Component Usage						
	Component	Raw	Utilized	w/ECC	AVF 0.5	AVF 0.25
1	Configuration (CRAM)	8486	3817	3817	1909	955
1	--Logic	2125	1071	1071	536	268
2	--Routing	6314	2716	2716	1358	679
3	--I/O config	47	30	30	15	8
2	RAM	41696	8593	1218	609	304
1	--HARD-IP (E.G. PCIe)	1446	692	0	0	0
2	--Embedded RAM	40250	7901	1218	609	304
3	--MLAB (LUTRAM)	0	0	0	0	0
3	Registers	2563	794	794	396	198
1	--Hard-IP FF	474	177	177	88	44
2	--DSP/M20K FF	298	61	61	30	15
3	--Design FF	1791	556	556	278	139
4						
5	TOTAL	52745	13204	5829	2914	1457

### 16.7.2.1 Component FIT Rates

The Projected SEU FIT by Component report shows FIT for the following components:

- SRAM embedded memory in embedded processors hard IP and M20K or M10K blocks
- CRAM used for LUT masks and routing configuration bits
- LABs in MLAB mode
- I/O configuration registers, which the FPGA implements differently than CRAM and design flipflops
- Standard flipflops the design uses in the address and data registers of M20K blocks, in DSP blocks, and in hard IP
- User flipflops the design implements in logic cells (ALMs or LEs)

### 16.7.2.2 Raw FIT

The Intel Quartus Prime Projected SEU FIT by Component Usage report provides raw FIT data. Raw FIT is the FIT rate of the FPGA if the design uses every component. Raw FIT data is not design specific.

*Note:* The Intel Reliability Report, available on the Altera web site, also provides reliability data and testing procedures for Intel FPGA devices.



The Intel Quartus Prime software computes the FIT for each component using (component Mb × intrinsic FIT/Mb × Neutron Flux Multiplier) for the device family and process node. (For flip flops, “Mb” represents a million flip flops.)

To give the worst-case raw FIT, the report assumes the maximum amount of CRAM that implements MLABs in the device. Thus, the CRAM raw FIT is the sum of CRAM and MLAB entries.

*Note:* The Intel Quartus Prime software counts device bits for target devices using different parameter information than the Reliability Report. Therefore, expect a ±5% variation in the Projected SEU FIT by Component Usage report **Raw** column compared to the Reliability Report data.

### Related Links

[Intel FPGA Reliability Report](#)

## 16.7.2.3 Utilized FIT

The **Utilized** column shows FIT calculations considering only resources that the design actually uses. Since SEU events in unused resources do not affect the FPGA, you can safely ignore these bits for resiliency statistics.

Additionally, the **Utilized** column discounts unused memory bits. For example, implementing a 16 × 16 memory in an M20K block uses only 256 bits of the 20 Kb.

*Note:* The Error Detection flag and the Projected SEU FIT by Component report do not distinguish between critical bit upsets, such as fundamental control logic, or non critical bit upsets, such as initialization logic that executes only once in the design. Apply hierarchy tags at the system level to filter out less important logic errors.

The Projected SEU FIT by Component report's **Utilized** CRAM FIT represents provable deflation of the FIT rate to account for CRAM upsets that do not matter to the design. Thus, the SEU incidence is always higher than the utilized FIT rate.

### Related Links

[Designating the Sensitivity of the Design Hierarchy](#) on page 1007

### 16.7.2.3.1 Comparing .smh Critical Bits Report to Utilized Bit Count

The number of design critical bits that the Compiler reports during .smh generation correlates to the utilized bits in the report, but it is not the same value. The difference occurs because the .smh file includes all bits in a resource, even when the resource usage is partial.

### 16.7.2.3.2 Considerations for Small Designs

The raw FIT for the entire device is always correct. In contrast, the utilized FIT is very conservative, and only becomes accurate for designs that reasonably fill up the chosen device. FPGAs contain overhead, such as the configuration state machine, the clock network control logic, and the I/O calibration block. These infrastructure blocks contain flip flops, memories, and sometimes I/O configuration blocks.

The Projected SEU FIT by Component report includes the constant overhead for GPIO and HSSI calibration circuitry for first I/O block or transceiver the design uses. Because of this overhead, the FIT of a 1-transceiver design is much higher than 1/10



the FIT of a 10-transceiver design. However, a trivial design such as “a single AND gate plus flipflop” could use so few bits that its CRAM FIT rate is 0.01, which the report rounds to zero.

#### 16.7.2.4 Mitigated FIT

You can lower FIT by reducing the observed FIT rate, such as by enabling ECC. You can also use the optional M20K ECC to mitigate FIT, as well as the (not optional) hard processor ECC and other hard IP such as memory controllers, PCIe, and I/O calibration blocks.

The Projected SEU FIT by Component Usage report's **w/ECC** column represents the FPGA's lowest guaranteed, provable FIT rate that the Intel Quartus Prime software can calculate. ECC does not affect CRAM and flipflop rates; therefore, the data in the **w/ECC** column for these components is the same as the in **Utilized** column.

The ECC code strength varies with the device family. In Intel Arria 10 devices, the M20K block can correct up to two errors, and the FIT rate beyond two (not corrected) is small enough to be negligible in the total.

An MLAB is simply a LAB configured with writable CRAM. However, when the Intel Quartus Prime software configures the RAM as write enabled (MLAB), the MLAB has a slightly different FIT/Mb. The Projected SEU FIT by Component Usage report displays a FIT rate in the MLAB row when the design uses MLABs, otherwise the report accounts for the block's FIT in the CRAM row. During compilation, if the Intel Quartus Prime software changes a LAB to an MLAB, the FIT accounting moves from the LAB row to the MLAB row.

The **w/ECC** column does not account for other forms of FIT protection in the design, such as designer-inserted parity, soft ECC blocks, bounds checking, system monitors, triple-module redundancy, or the impact of higher-level protocols on general fault tolerance. Additionally, it does not account for single event effects that occur in the logic but the design never reads or notices. For example, if you implement a non-ECC FIFO function 512 bits deep and an SEU event occurs outside of the front and back pointers, the application does not observe the SEU event. However, the report accounts for the full 512 bit deep memory and includes it in the **w/ECC** FIT rate. Designers often combine these factors into general deflation factors (called architectural vulnerability factors or AVF) based on knowledge of their design. Designers use AVF factors as low (aggressive) as 5% and as high (conservative) as 50% based on experience, fault-injection or neutron beam testing, or high-level system monitors.

#### 16.7.2.5 Architectural Vulnerability Factor

The Single Event Functional Interrupt (SEFI) ratio measures bit errors due to SEU strikes versus functional interrupts. Minimizing this ratio improves SEU mitigation. 10% SEFI factors are a typical specification to deflate the raw FIT to that observed in practice. For reference, the last two columns in the Projected SEU FIT by Component Usage report show AVF deflations for a conservative SEFI of 50% and a moderate SEFI of 25%.

SEFI represents a combination of factors. A utilization + ECC factor of 40% and AVF of 25% thus represents a global SEFI factor of 10%, because  $0.4 \times 0.25 = 0.1$ . An end-to-end SEFI factor of 10% is typical for a full design.

## Related Links

[Understanding Single Event Functional Interrupts in FPGA Designs White Paper](#)

### 16.7.3 Enabling the Projected SEU FIT by Component Usage Report

The Intel Quartus Prime Fitter generates the Projected SEU FIT by Component Usage report. The Intel Quartus Prime software only generates reports for designs that successfully pass place and route.

To enable the report:

1. Obtain and install the SEU license.
2. Add the following assignments to your project's .qsf file:

```
set_global_assignment -name ENABLE_ADV_SEU_DETECTION ON  
set_global_assignment -name SEU_FIT_REPORT ON
```

### 16.8 Triple-Module Redundancy

Use Triple-Module Redundancy (TMR) if your system cannot suffer downtime due to SEU. TMR is an established SEU mitigation technique for improving hardware fault tolerance. A TMR design has three identical instances of hardware with voting hardware at the output. If an SEU affects one of the hardware instances, the voting logic notes the majority output. This operation masks malfunctioning hardware.

With TMR, your design does not suffer downtime in the case of a single SEU; if the system detects a faulty module, the system can scrub the error by reprogramming the module. The error detection and correction time is many orders of magnitude less than the MTBF of SEU events. Therefore, the system can repair a soft interrupt before another SEU affects another instance in the TMR application.

The disadvantage of TMR is its hardware resource cost: it requires three times as much hardware in addition to voting logic. You can minimize this hardware cost by implementing TMR for only the most critical parts of your design.

There are several automated ways to generate TMR designs by automatically replicating designated functions and synthesizing the required voting logic. Synopsys offers automated TMR synthesis.

### 16.9 Evaluating a System's Response to Functional Upsets

SEUs can strike any memory element, so you must test the system to ensure a comprehensive recovery response. The Intel Quartus Prime software includes the Fault Injection Debugger to aid in SEU recovery. You can use the Fault Injection Debugger graphically with the GUI, or you can use command line assignments.

In Intel Arria 10 and Intel Cyclone 10 GX designs, the Fault Injection Debugger works together with the Fault Injection IP core. To use the debugging feature you must instantiate the Fault Injection IP core in the FPGA design. During debugging, the IP core flips a CRAM bit by dynamically reconfiguring the frame containing the CRAM bit.

**Note:** Intel Stratix 10 devices do not require instantiation of the Fault Injection IP Core.



With the Fault Injection Debugger, you can operate the FPGA in the system and inject random CRAM bit flips. These simulated SEU strikes allow you to observe how the FPGA and the system detect and recover from SEUs. Depending on the results, you can refine the system's recovery sequence.

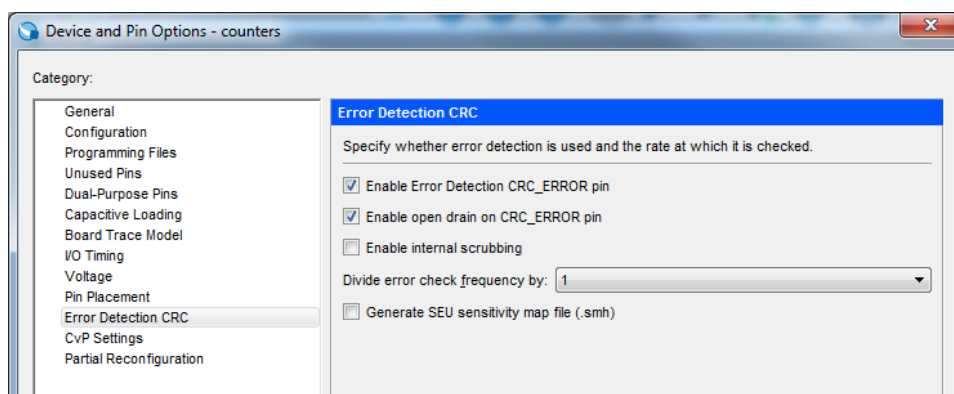
### Related Links

- [Intel FPGA Fault Injection IP Core User Guide](#)
- [Debugging Single Event Upsets Using the Fault Injection Debugger](#)  
In *Intel Quartus Prime Pro Edition Handbook Volume 3*

## 16.10 CRAM Error Detection Settings Reference

To define these settings in the Intel Quartus Prime software, point to **Assignments** ► **Device** ► **Device and Pin Options** ► **Error Detection CRC**.

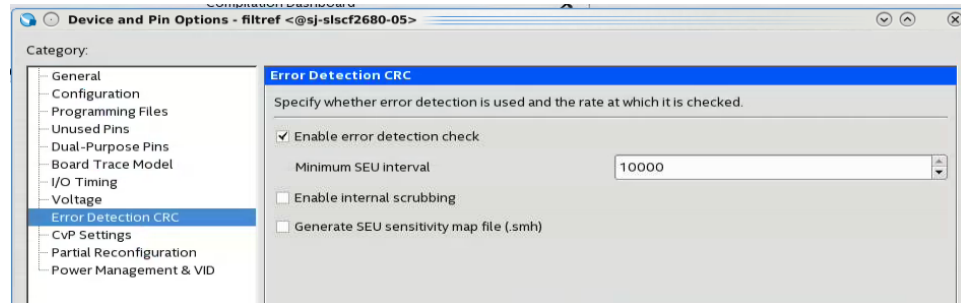
**Figure 327. Device and Pin Options Error Detection CRC Tab (Intel Arria 10 and Intel Cyclone 10 GX devices)**



**Table 269. CRC Errors Settings (Intel Arria 10 and Intel Cyclone 10 GX devices)**

Setting	Description
<b>Enable Error Detection CRC_ERROR pin</b>	Enables CRAM frame scanning
<b>Enable open drain on CRC_ERROR pin</b>	Enables the CRC_ERROR pin as an open-drain output
<b>Divide error check frequency by</b>	To guarantee the availability of a clock, the EDCRC function operates on an independent clock generated internally on the FPGA itself. To enable EDCRC operation on a divided version of the clock, select a value from the list.

**Figure 328. Device and Pin Options Error Detection CRC Pane (Intel Stratix 10 devices)**



**Table 270. CRC Errors Settings (Intel Stratix 10 devices)**

Setting	Description
<b>Enable error detection check</b>	If turned on, the device checks the validity of the programming data in the device. Any changes in the data while the device is in operation generates an error. The status is SEU_ERROR output SDM_IO.
<b>Minimum SEU interval</b>	Specifies the minimum time between two checks of the same bit. Setting to 0 means check as frequently as possible. Setting to a large value saves power. The unit of interval is millisecond. The maximum allowed number of intervals is 10000.
<b>Enable internal scrubbing</b>	If enabled, corrects single error or double adjacent error within the core configuration memory while the device is still running.

## 16.11 Document Revision History

**Table 271. Document Revision History**

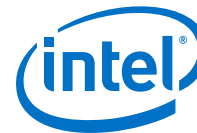
Date	Version	Changes
2017.12.15	17.1.0	<ul style="list-style-type: none"> <li>Added information about how the CRAM error detection works on Intel Stratix 10 devices.</li> <li>Clarified the type of errors that the Internal Scrubbing feature supports for Intel Stratix 10 devices.</li> <li>Separated description of the SEU FIT Parameters Report and the Projected SEU FIT by Component Usage Report.</li> </ul>
2017.11.06	17.1.0	<ul style="list-style-type: none"> <li>Added support for Internal Scrubbing on Intel Stratix 10 devices.</li> <li>Added support for Hierarchy Tagging.</li> <li>Added topic: CRAM Error Detection Settings Reference.</li> <li>Removed topic: Scanning CRAM Frames.</li> </ul>
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Implemented Intel rebranding.</li> <li>Removed incorrect link to Reliability Report.</li> <li>Added MAX 10 and Cylone IV to list of devices supporting Projected SEU FIT by Component Usage Report.</li> </ul>
2016.05.24	16.0.1	<ul style="list-style-type: none"> <li>Corrected the steps to enable the SEU FIT reports.</li> </ul>
2016.05.03	16.0.0	<ul style="list-style-type: none"> <li>Documented the new SEU FIT reports.</li> <li>Inconsequential wording changes for conformance to style.</li> </ul>

*continued...*





Date	Version	Changes
2015.11.02	15.1.0	<ul style="list-style-type: none"><li>• Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li></ul>
June 2014	2014.06.30	<ul style="list-style-type: none"><li>• Updated formatting.</li><li>• Added "Mitigating SEU Effects in Embedded User RAM" section.</li><li>• Added "Altera Advanced SEU Detection IP Core" section.</li></ul>
November 2012	2012.11.01	<ul style="list-style-type: none"><li>• Preliminary release.</li></ul>



## 17 Optimizing the Design Netlist

---

This chapter describes how you can use the Intel Quartus Prime Netlist Viewers to analyze and debug your designs.

As FPGA designs grow in size and complexity, the ability to analyze, debug, optimize, and constrain your design is critical. With today's advanced designs, several design engineers are involved in coding and synthesizing different design blocks, making it difficult to analyze and debug the design. The Intel Quartus Prime RTL Viewer and Technology Map Viewer provide powerful ways to view your initial and fully mapped synthesis results during the debugging, optimization, and constraint entry processes.

### Related Links

- [When to Use the Netlist Viewers: Analyzing Design Problems](#) on page 1018
- [Introduction to the User Interface](#) on page 1022
- [Intel Quartus Prime Design Flow with the Netlist Viewers](#) on page 1019
- [RTL Viewer Overview](#) on page 1020
- [Technology Map Viewer Overview](#) on page 1021
- [Filtering in the Schematic View](#) on page 1031
- [Cross-Probing to a Source Design File and Other Intel Quartus Prime Windows](#) on page 1036
- [Cross-Probing to the Netlist Viewers from Other Intel Quartus Prime Windows](#) on page 1037
- [Viewing a Timing Path](#) on page 1037

### 17.1 When to Use the Netlist Viewers: Analyzing Design Problems

You can use the Netlist Viewers to analyze and debug your design. The following simple examples show how to use the RTL Viewer and Technology Map Viewer to analyze problems encountered in the design process.

Using the RTL Viewer is a good way to view your initial synthesis results to determine whether you have created the necessary logic, and that the logic and connections have been interpreted correctly by the software. You can use the RTL Viewer to check your design visually before simulation or other verification processes. Catching design errors at this early stage of the design process can save you valuable time.

If you see unexpected behavior during verification, use the RTL Viewer to trace through the netlist and ensure that the connections and logic in your design are as expected. Viewing your design helps you find and analyze the source of design problems. If your design looks correct in the RTL Viewer, you know to focus your analysis on later stages of the design process and investigate potential timing violations or issues in the verification flow itself.



You can use the Technology Map Viewer to look at the results at the end of Analysis and Synthesis. If you have compiled your design through the Fitter stage, you can view your post-mapping netlist in the Technology Map Viewer (Post-Mapping) and your post-fitting netlist in the Technology Map Viewer. If you perform only Analysis and Synthesis, both the Netlist Viewers display the same post-mapping netlist.

In addition, you can use the RTL Viewer or Technology Map Viewer to locate the source of a particular signal, which can help you debug your design. Use the navigation techniques described in this chapter to search easily through your design. You can trace back from a point of interest to find the source of the signal and ensure the connections are as expected.

The Technology Map Viewer can help you locate post-synthesis nodes in your netlist and make assignments when optimizing your design. This functionality is useful when making a multicycle clock timing assignment between two registers in your design. Start at an I/O port and trace forward or backward through the design and through levels of hierarchy to find nodes of interest, or locate a specific register by visually inspecting the schematic.

Throughout your FPGA design, debug, and optimization stages, you can use all of the netlist viewers in many ways to increase your productivity while analyzing a design.

#### Related Links

- [Intel Quartus Prime Design Flow with the Netlist Viewers](#) on page 1019
- [RTL Viewer Overview](#) on page 1020
- [Technology Map Viewer Overview](#) on page 1021

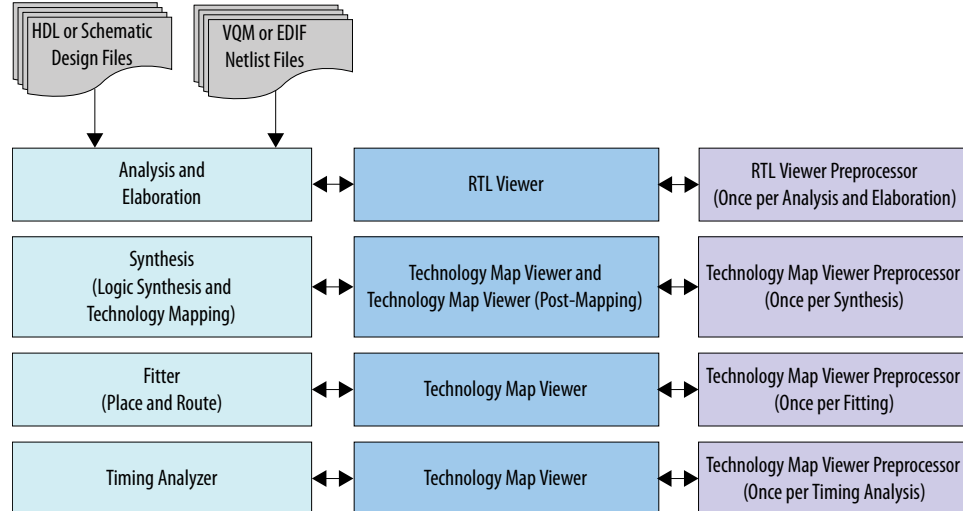
## 17.2 Intel Quartus Prime Design Flow with the Netlist Viewers

When you first open one of the Netlist Viewers after compiling the design, a preprocessor stage runs automatically before the Netlist Viewer opens.

Click the link in the preprocessor process box to go to the **Settings > Compilation Process Settings** page where you can turn on the **Run Netlist Viewers preprocessing during compilation** option. If you turn this option on, the preprocessing becomes part of the full project compilation flow and the Netlist Viewer opens immediately without displaying the preprocessing dialog box.

**Figure 329. Intel Quartus Prime Design Flow Including the RTL Viewer and Technology Map Viewer**

This figure shows how Netlist Viewers fit into the basic Intel Quartus Prime design flow.



Before the Netlist Viewer can run the preprocessor stage, you must compile your design:

- To open the RTL Viewer first perform Analysis and Elaboration.
- To open the Technology Map Viewer (Post-Fitting) or the Technology Map Viewer (Post-Mapping), first perform Analysis and Synthesis.

The Netlist Viewers display the results of the last successful compilation.

- Therefore, if you make a design change that causes an error during Analysis and Elaboration, you cannot view the netlist for the new design files, but you can still see the results from the last successfully compiled version of the design files.
- If you receive an error during compilation and you have not yet successfully run the appropriate compilation stage for your project, the Netlist Viewer cannot be displayed; in this case, the Intel Quartus Prime software issues an error message when you try to open the Netlist Viewer.

**Note:** If the Netlist Viewer is open when you start a new compilation, the Netlist Viewer closes automatically. You must open the Netlist Viewer again to view the new design netlist after compilation completes successfully.

### 17.3 RTL Viewer Overview

The RTL Viewer allows you to view a register transfer level (RTL) graphical representation of your Intel Quartus Prime Pro Edition synthesis results or your third-party netlist file in the Intel Quartus Prime software.

You can view results after Analysis and Elaboration when your design uses any supported Intel Quartus Prime design entry method, including Verilog HDL Design Files (.v), SystemVerilog Design Files (.sv), VHDL Design Files (.vhdl), AHDL Text Design Files (.tdf), or schematic Block Design Files (.bdf). You can also view the hierarchy



of atom primitives (such as device logic cells and I/O ports) when your design uses a synthesis tool to generate a Verilog Quartus Mapping File (.vqm) or Electronic Design Interchange Format (.edf) file.

The RTL Viewer displays a schematic view of the design netlist after Analysis and Elaboration or netlist extraction is performed by the Intel Quartus Prime software, but before technology mapping and any synthesis or fitter optimizations. This view a preliminary pre-optimization design structure and closely represents your original source design.

- If you synthesized your design with the Intel Quartus Prime Pro Edition synthesis, this view shows how the Intel Quartus Prime software interpreted your design files.
- If you use a third-party synthesis tool, this view shows the netlist written by your synthesis tool.

While displaying your design, the RTL Viewer optimizes the netlist to maximize readability:

- Removes logic with no fan-out (unconnected output) or fan-in (unconnected inputs) from the display.
- Hides default connections such as  $V_{CC}$  and GND.
- Groups pins, nets, wires, module ports, and certain logic into buses where appropriate.
- Groups constant bus connections are grouped.
- Displays values in hexadecimal format.
- Converts NOT gates into bubble inversion symbols in the schematic.
- Merges chains of equivalent combinational gates into a single gate; for example, a 2-input AND gate feeding a 2-input AND gate is converted to a single 3-input AND gate.

To run the RTL Viewer for a Intel Quartus Prime project, first analyze the design to generate an RTL netlist. To analyze the design and generate an RTL netlist, click **Processing > Start Analysis & Elaboration**. You can also perform a full compilation on any process that includes the initial Analysis and Elaboration stage of the Intel Quartus Prime compilation flow.

To open the RTL Viewer, click **Tools > Netlist Viewers RTL Viewer**.

### Related Links

[Introduction to the User Interface](#) on page 1022

## 17.4 Technology Map Viewer Overview

The Intel Quartus Prime Technology Map Viewer provides a technology-specific, graphical representation of your design after Analysis and Synthesis or after the Fitter has mapped your design into the target device.

The Technology Map Viewer shows the hierarchy of atom primitives (such as device logic cells and I/O ports) in your design. For supported device families, you can also view internal registers and look-up tables (LUTs) inside logic cells (LCELLs), and registers in I/O atom primitives.



Where possible, the Intel Quartus Prime software maintains the port names of each hierarchy throughout synthesis. However, the software may change or remove port names from the design. For example, if a port is unconnected or driven by GND or  $V_{CC}$ , the software removes it during synthesis. If a port name changes, the software assigns a related user logic name in the design or a generic port name such as IN1 or OUT1.

You can view your Intel Quartus Prime technology-mapped results after synthesis, fitting, or timing analysis. To run the Technology Map Viewer for a Intel Quartus Prime project, on the Processing menu, point to **Start** and click **Start Analysis & Synthesis** to synthesize and map the design to the target technology. At this stage, the Technology Map Viewer shows the same post-mapping netlist as the Technology Map Viewer (Post-Mapping). You can also perform a full compilation, or any process that includes the synthesis stage in the compilation flow.

If you have completed the Fitter stage, the Technology Map Viewer shows the changes made to your netlist by the Fitter, such as physical synthesis optimizations, while the Technology Map Viewer (Post-Mapping) shows the post-mapping netlist. If you have completed the Timing Analysis stage, you can locate timing paths from the Timing Analyzer report in the Technology Map Viewer.

To open the Technology Map Viewer, on the Tools menu, point to **Netlist Viewers** and click **Technology Map Viewer (Post-Fitting)** or **Technology Map Viewer (Post Mapping)**.

#### Related Links

- [View Contents of Nodes in the Schematic View](#) on page 1032
- [Viewing a Timing Path](#) on page 1037
- [Introduction to the User Interface](#) on page 1022

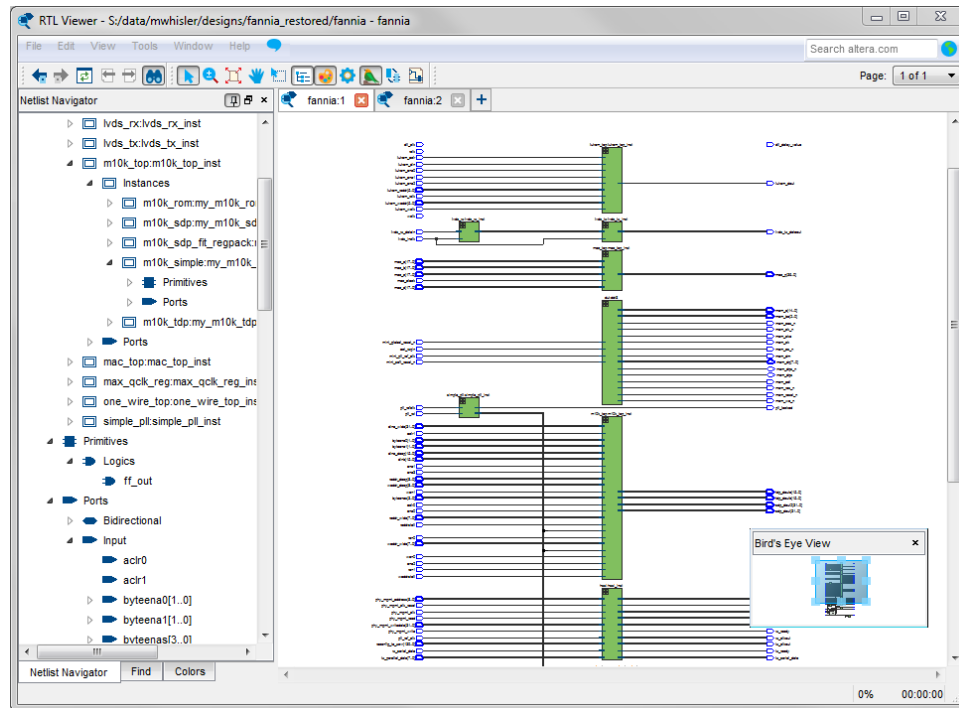
## 17.5 Introduction to the User Interface

The Netlist Viewer is a graphical user-interface for viewing and manipulating nodes and nets in the netlist.

The RTL Viewer and Technology Map Viewer each consist of these main parts:

- The **Netlist Navigator** pane—displays a representation of the project hierarchy.
- The **Find** pane—allows you to find and locate specific design elements in the schematic view.
- The **Properties** pane displays the properties of the selected block when you select **Properties** from the shortcut menu.
- The schematic view—displays a graphical representation of the internal structure of your design.

Figure 330. RTL Viewer

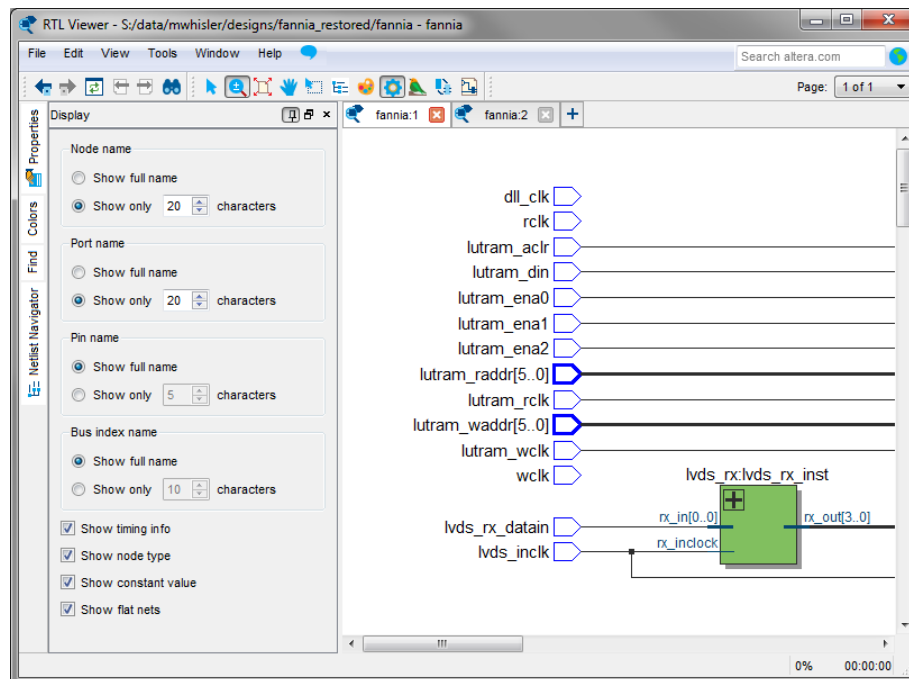


Netlist Viewers also contain a toolbar that provides tools to use in the schematic view.

- Use the **Back** and **Forward** buttons to switch between schematic views. You can go forward only if you have not made any changes to the view since going back. These commands do not undo an action, such as selecting a node. The Netlist Viewer caches up to ten actions including filtering, hierarchy navigation, netlist navigation, and zoom actions.
- The **Refresh** button to restore the schematic view and optimizes the layout. **Refresh** does not reload the database if you change your design and recompile.
- Click the **Find** button opens and closes the **Find** pane.
- Click the **Selection Tool** and **Zoom Tool** buttons to toggle between the selection mode and zoom mode.
- Click the **Fit in Page** button resets the schematic view to encompass the entire design.
- Use the **Hand Tool** to change the focus of the viewer without changing the perspective.
- Click the **Area Selection Tool** to drag a selection box around ports, pins, and nodes in an area.
- Click the **Netlist Navigator** button to open or close the **Netlist Navigator** pane.
- Click the **Color Settings** button to open the **Colors** pane where you can customize the Netlist Viewer color scheme.

- Click the **Display Settings** button to open the **Display** pane where you can specify the following settings:
  - **Show full name** or **Show only <n> characters**. You can specify this separately for **Node name**, **Port name**, **Pin name**, or **Bus name**.
  - Turn **Show timing info** on or off.
  - Turn **Show node type** on or off.
  - Turn **Show constant value** on or off.
  - Turn **Show flat nets** on or off.

Figure 331. Display Settings



- The **Bird's Eye View** button opens the **Bird's Eye View** window which displays a miniature version of your design and allows you to navigate within the design and adjust the magnification in the schematic view quickly.
- The **Show/Hide Instance Pins** button can toggle the display of instance pins not displayed by functions such as cross-probing between a Netlist Viewer and Timing Analyzer. You can also use it to hide unconnected instance pins when filtering a node results in large numbers of unconnected or unused pins. Instance pins are hidden by default.
- The **Show Netlist on One Page** button displays the netlist on a single page if the Netlist Viewer has split the design across several pages. This can make netlist tracing easier.

You can have only one RTL Viewer, one Technology Map Viewer (Post-Fitting), and one Technology Map Viewer (Post-Mapping) window open at the same time, although each window can show multiple pages, each with multiple tabs. For example, you cannot have two RTL Viewer windows open at the same time.





### Related Links

- [RTL Viewer Overview](#) on page 1020
- [Technology Map Viewer Overview](#) on page 1021
- [Netlist Navigator Pane](#) on page 1025
- [Netlist Viewers Find Pane](#) on page 1027
- [Properties Pane](#) on page 1025

## 17.5.1 Netlist Navigator Pane

The **Netlist Navigator** pane displays the entire netlist in a tree format based on the hierarchical levels of the design. In each level, similar elements are grouped into subcategories.

You can use the **Netlist Navigator** pane to traverse through the design hierarchy to view the logic schematic for each level. You can also select an element in the **Netlist Navigator** to highlight in the schematic view.

*Note:* Nodes inside atom primitives are not listed in the **Netlist Navigator** pane.

For each module in the design hierarchy, the **Netlist Navigator** pane displays the applicable elements listed in the following table. Click the “+” icon to expand an element.

**Table 272. Netlist Navigator Pane Elements**

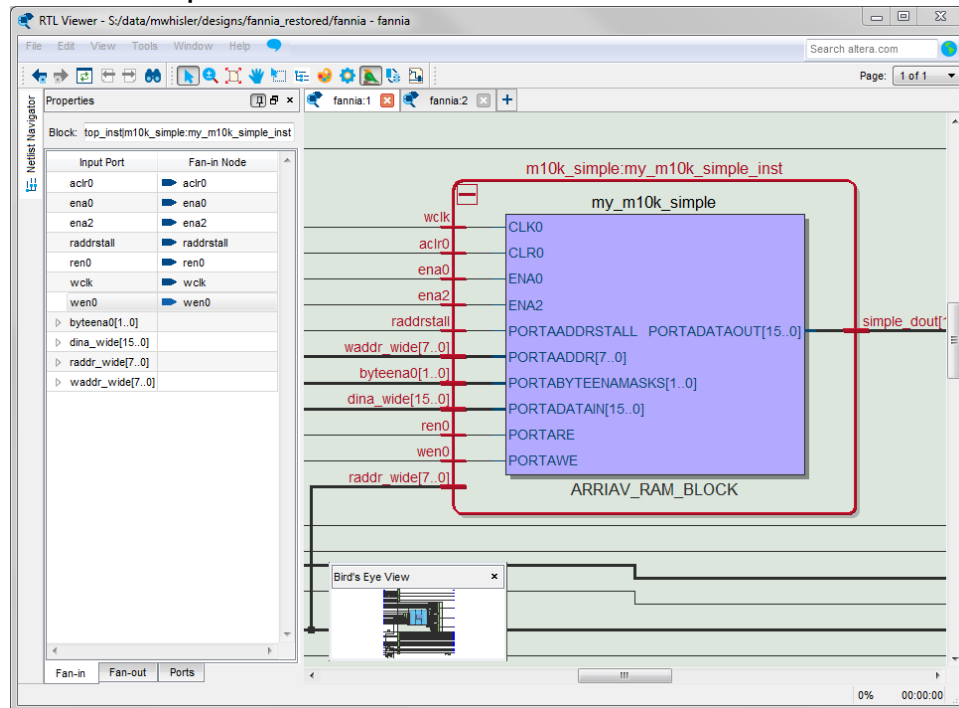
Elements	Description
Instances	Modules or instances in the design that can be expanded to lower hierarchy levels.
Primitives	Low-level nodes that cannot be expanded to any lower hierarchy level. These primitives include: <ul style="list-style-type: none"><li>• Registers and gates that you can view in the RTL Viewer when using Intel Quartus Prime Pro Edition synthesis.</li><li>• Logic cell atoms in the Technology Map Viewer or in the RTL Viewer when using a VQM or EDIF from third-party synthesis software</li></ul> In the Technology Map Viewer, you can view the internal implementation of certain atom primitives, but you cannot traverse into a lower-level of hierarchy.
Ports	The I/O ports in the current level of hierarchy. <ul style="list-style-type: none"><li>• Pins are device I/O pins when viewing the top hierarchy level and are I/O ports of the design when viewing the lower-levels.</li><li>• When a pin represents a bus or an array of pins, expand the pin entry in the list view to see individual pin names.</li></ul>

## 17.5.2 Properties Pane

You can view the properties of an instance or primitive using the **Properties** pane.

**Figure 332. Properties Pane**

To view the properties of an instance or primitive in the RTL Viewer or Technology Map Viewer, right-click the node and click **Properties**.



The **Properties** pane contains tabs with the following information about the selected node:

- The **Fan-in** tab displays the **Input port** and **Fan-in Node**.
- The **Fan-out** tab displays the **Output port** and **Fan-out Node**.
- The **Parameters** tab displays the **Parameter Name** and **Values** of an instance.
- The **Ports** tab displays the **Port Name** and **Constant** value (for example,  $V_{CC}$  or GND). The possible value of a port are listed below.

**Table 273. Possible Port Values**

Value	Description
$V_{CC}$	The port is not connected and has $V_{CC}$ value (tied to $V_{CC}$ )
GND	The port is not connected and has GND value (tied to GND)
--	The port is connected and has value (other than $V_{CC}$ or GND)
Unconnected	The port is not connected and has no value (hanging)

If the selected node is an atom primitive, the **Properties** pane displays a schematic of the internal logic.



### 17.5.3 Netlist Viewers Find Pane

You can narrow the range of the search process by setting the following options in the **Find** pane:

- Click **Browse** in the **Find** pane to specify the hierarchy level of the search. In the **Select Hierarchy Level** dialog box, select the particular instance you want to search.
- Turn on the **Include subentities** option to include child hierarchies of the parent instance during the search.
- Click **Options** to open the **Find Options** dialog box. Turn on **Instances**, **Nodes**, **Ports**, or any combination of the three to further refine the parameters of the search.

When you click the **List** button, a progress bar appears below the **Find** box.

All results that match the criteria you set are listed in a table. When you double-click an item in the table, the related node is highlighted in red in the schematic view.

## 17.6 Schematic View

The schematic view is shown on the right side of the RTL Viewer and Technology Map Viewer. The schematic view contains a schematic representing the design logic in the netlist. This view is the main screen for viewing your gate-level netlist in the RTL Viewer and your technology-mapped netlist in the Technology Map Viewer.

The RTL Viewer and Technology Map Viewer attempt to display schematic in a single page view by default. If the schematic crosses over to several pages, you can highlight a net and use connectors to trace the signal in a single page.

### 17.6.1 Display Schematics in Multiple Tabbed View

The RTL Viewer and Technology Map Viewer support multiple tabbed views.

With multiple tabbed view, schematics can be displayed in different tabs. Selection is independent between tabbed views, but selection in the tab in focus is synchronous with the Netlist Navigator pane.

To create a new blank tab, click the **New Tab** button at the end of the tab row . You can now drag a node from the **Netlist Navigator** pane into the schematic view.

Right-click in a tab to see a shortcut menu to perform the following actions:

- Create a blank view with **New Tab**
- Create a **Duplicate Tab** of the tab in focus
- Choose to **Cascade Tabs**
- Choose to **Tile Tabs**
- Choose **Close Tab** to close the tab in focus
- Choose **Close Other Tabs** to close all tabs except the tab in focus

## 17.6.2 Schematic Symbols

The symbols for nodes in the schematic represent elements of your design netlist. These elements include input and output ports, registers, logic gates, Intel primitives, high-level operators, and hierarchical instances.

**Note:** The logic gates and operator primitives appear only in the RTL Viewer. Logic in the Technology Map Viewer is represented by atom primitives, such as registers and LCELLs.

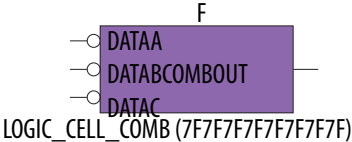
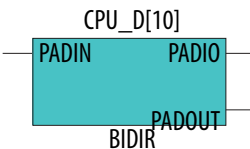
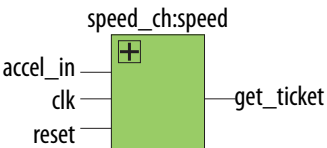
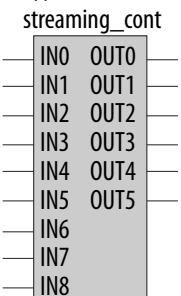
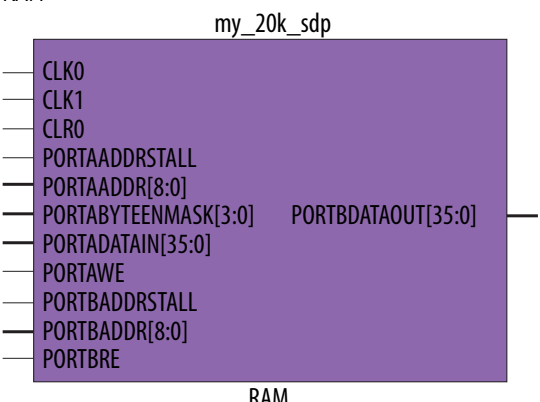
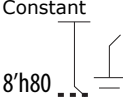
**Table 274. Symbols in the Schematic View**

This table lists and describes the primitives and basic symbols that you can display in the schematic view of the RTL Viewer and Technology Map Viewer.

Symbol	Description
<p>I/O Ports</p>	<p>An input, output, or bidirectional port in the current level of hierarchy. A device input, output, or bidirectional pin when viewing the top-level hierarchy. The symbol can also represent a bus. Only one wire is shown connected to the bidirectional symbol, representing the input and output paths.</p> <p>Input symbols appear on the left-most side of the schematic. Output and bidirectional symbols appear on the right-most side of the schematic.</p>
<p>I/O Connectors</p>	<p>An input or output connector, representing a net that comes from another page of the same hierarchy. To go to the page that contains the source or the destination page, double-click the connector to jump to the appropriate page.</p>
<p>OR, AND, XOR Gates</p>	<p>An OR, AND, or XOR gate primitive (the number of ports can vary). A small circle (bubble symbol) on an input or output port indicates the port is inverted.</p>
<p>MULTIPLEXER</p>	<p>A multiplexer primitive with a selector port that selects between port 0 and port 1. A multiplexer with more than two inputs is displayed as an operator.</p>
<p>BUFFER</p>	<p>A buffer primitive. The figure shows the tri-state buffer, with an inverted output enable port. Other buffers without an enable port include LCELL, SOFT, CARRY, and GLOBAL. The NOT gate and EXP expander buffers use this symbol without an enable port and with an inverted output port.</p>
<p>LATCH</p>	<p>A latch/DFF (data flipflop) primitive. A DFF has the same ports as a latch and a clock trigger. The other flipflop primitives are similar:</p> <ul style="list-style-type: none"> <li>• DFFEA (data flipflop with enable and asynchronous load) primitive with additional ALOAD asynchronous load and ADATA data signals</li> <li>• DFFEAS (data flipflop with enable and synchronous and asynchronous load), which has ASDATA as the secondary data port</li> </ul>
<p>Atom Primitive</p>	<p>An atom primitive. The symbol displays the atom name, the port names, and the atom type. The blue shading indicates an atom primitive for which you can view the internal details.</p>

*continued...*



Symbol	Description
 <p>LOGIC_CELL_COMB (7F7F7F7F7F7F7F7F)</p>	
<p>Other Primitive</p>  <p>CPU_D[10]</p>	<p>Any primitive that does not fall into the previous categories. Primitives are low-level nodes that cannot be expanded to any lower hierarchy. The symbol displays the port names, the primitive or operator type, and its name.</p>
<p>Instance</p>  <p>speed_ch:speed</p>	<p>An instance in the design that does not correspond to a primitive or operator (a user-defined hierarchy block). The symbol displays the port name and the instance name.</p>
<p>Encrypted Instance</p>  <p>streaming_cont</p>	<p>A user-defined encrypted instance in the design. The symbol displays the instance name. You cannot open the schematic for the lower-level hierarchy, because the source design is encrypted.</p>
<p>RAM</p>  <p>my_20k_sdp</p> <p>RAM</p>	<p>A synchronous memory instance with registered inputs and optionally registered outputs. The symbol shows the device family and the type of memory block. This figure shows a true dual-port memory block in a Stratix M-RAM block.</p>
<p>Constant</p>  <p>8'h80</p>	<p>A constant signal value that is highlighted in gray and displayed in hexadecimal format by default throughout the schematic.</p>

**Table 275. Operator Symbols in the RTL Viewer Schematic View**

The following lists and describes the additional higher level operator symbols in the RTL Viewer schematic view.

Symbol	Description
	An adder operator: $OUT = A + B$
	A multiplier operator: $OUT = A \times B$
	A divider operator: $OUT = A / B$
	Equals
	A left shift operator: $OUT = (A \ll COUNT)$
	A right shift operator: $OUT = (A \gg COUNT)$
	A modulo operator: $OUT = (A \% B)$
	A less than comparator: $OUT = (A < B : A > B)$
	A multiplexer: $OUT = DATA [SEL]$ The data range size is $2^{sel \text{ range size}}$
	A selector: A multiplexer with one-hot select input and more than two input signals
	A binary number decoder: $OUT = (\text{binary\_number}(IN) == x)$ for $x = 0$ to $x = 2^{(n+1)} - 1$



#### Related Links

- [Partition the Schematic into Pages](#) on page 1035
- [Follow Nets Across Schematic Pages](#) on page 1036

### 17.6.3 Select Items in the Schematic View

To select an item in the schematic view, ensure that the **Selection Tool** is enabled in the Netlist Viewer toolbar (this tool is enabled by default). Click an item in the schematic view to highlight it in red.

Select multiple items by pressing the Shift key while selecting with your mouse.

Items selected in the schematic view are automatically selected in the **Netlist Navigator** pane. The folder then expands automatically if it is required to show the selected entry; however, the folder does not collapse automatically when you are not using or you have deselected the entries.

When you select a hierarchy box, node, or port in the schematic view, the item is highlighted in red but none of the connecting nets are highlighted. When you select a net (wire or bus) in the schematic view, all connected nets are highlighted in red.

Once you have selected an item, you can perform different actions on it based on the contents of the shortcut menu which appears when you right-click your selection.

#### Related Links

[Netlist Navigator Pane](#) on page 1025

### 17.6.4 Shortcut Menu Commands in the Schematic View

When you right-click on an instance or primitive selected in the schematic view, the Netlist Viewer displays a shortcut menu.

If the selected item is a node, you see the following options:

- Click **Expand to Upper Hierarchy** to displays the parent hierarchy of the node in focus.
- Click **Copy ToolTip** to copy the selected item name to the clipboard. This command does not work on nets.
- Click **Hide Selection** to remove the selected item from the schematic view. This command does not delete the item from the design, merely masks it in the current view.
- Click **Filtering** to display a sub-menu with options for filtering your selection.

### 17.6.5 Filtering in the Schematic View

Filtering allows you to filter out nodes and nets in your netlist to view only the logic elements of interest to you.

You can filter your netlist by selecting hierarchy boxes, nodes, or ports of a node, that are part of the path you want to see. The following filter commands are available:

- **Sources**—Displays the sources of the selection.
- **Destinations**—Displays the destinations of the selection.
- **Sources & Destinations**—displays the sources and destinations of the selection.
- **Selected Nodes**—Displays only the selected nodes.
- **Between Selected Nodes**—Displays nodes and connections in the path between the selected nodes .
- **Bus Index**—Displays the sources or destinations for one or more indices of an output or input bus port .
- **Filtering Options**—Displays the **Filtering Options** dialog box:
  - **Stop filtering at register**—Turning this option on directs the Netlist Viewer to filter out to the nearest register boundary.
  - **Filter across hierarchies**—Turning this option on directs the Netlist Viewer to filter across hierarchies.
  - **Maximum number of hierarchy levels**—Sets the maximum number of hierarchy levels displayed in the schematic view.

To filter your netlist, select a hierarchy box, node, port, net, or state node, right-click in the window, point to **Filter** and click the appropriate filter command. The Netlist Viewer generates a new page showing the netlist that remains after filtering.

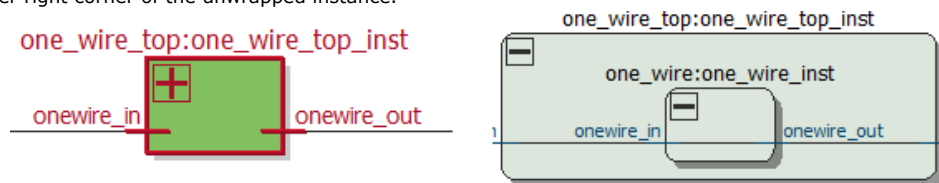
### 17.6.6 View Contents of Nodes in the Schematic View

In the RTL Viewer and the Technology Map Viewer, you can view the contents of nodes to see their underlying implementation details.

You can view LUTs, registers, and logic gates. You can also view the implementation of RAM and DSP blocks in certain devices in the RTL Viewer or Technology Map Viewer. In the Technology Map Viewer, you can view the contents of primitives to see their underlying implementation details.

**Figure 333. Wrapping and Unwrapping Objects**

If you can unwrap the contents of an instance, a plus symbol appears in the upper right corner of the object in the schematic view. To wrap the contents (and revert to the compact format), click the minus symbol in the upper right corner of the unwrapped instance.



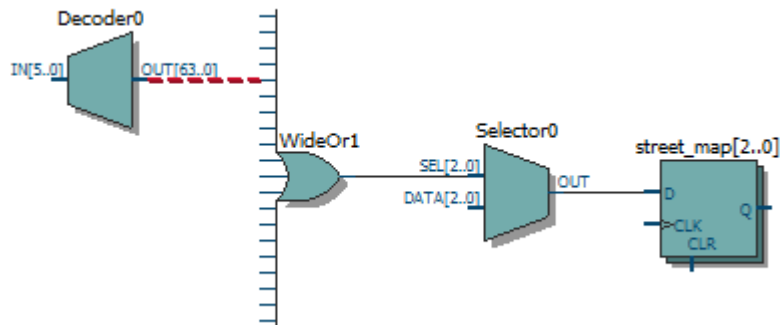
**Note:** In the schematic view, the internal details in an atom instance cannot be selected as individual nodes. Any mouse action on any of the internal details is treated as a mouse action on the atom instance.





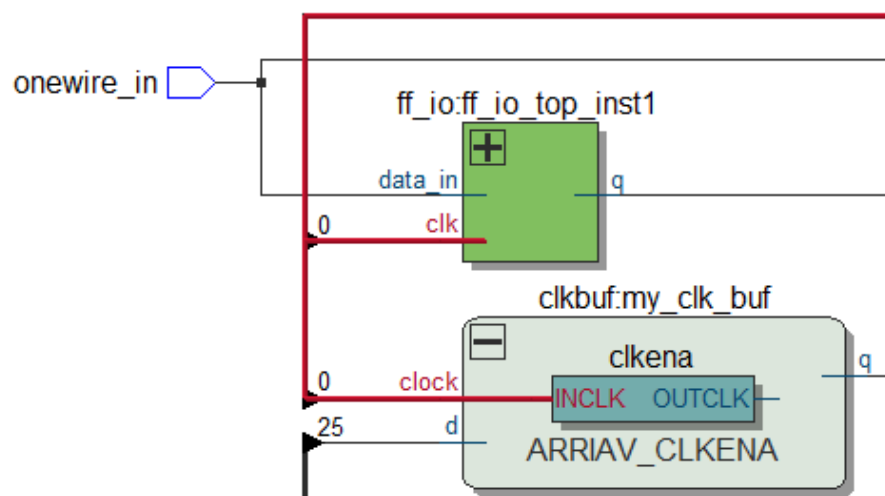
**Figure 334. Nodes with Connections Outside the Hierarchy**

In some cases, the selected instance connects to something outside the visible level of the hierarchy in the schematic view. In this case, the net appears as a dotted line. Double-click the dotted line to expand the view to display the destination of the connection .



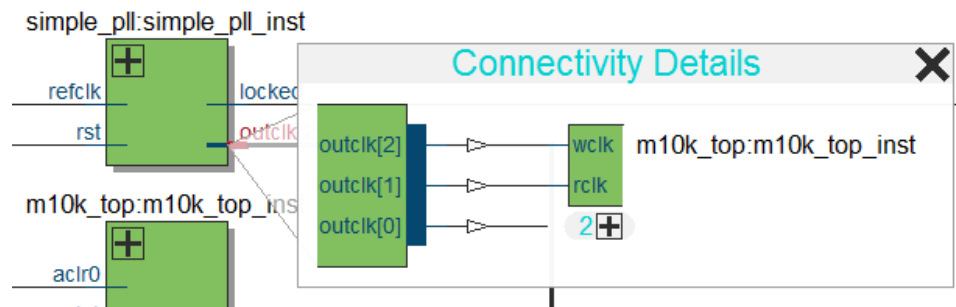
**Figure 335. Display Nets Across Hierarchies**

In cases where the net connects to an instance outside the hierarchy, you can select the net, and unwrap the node to see the destination ports.



**Figure 336. Show Connectivity Details**

You can select a bus port or bus pin and click **Connectivity Details** in the context menu for that object.



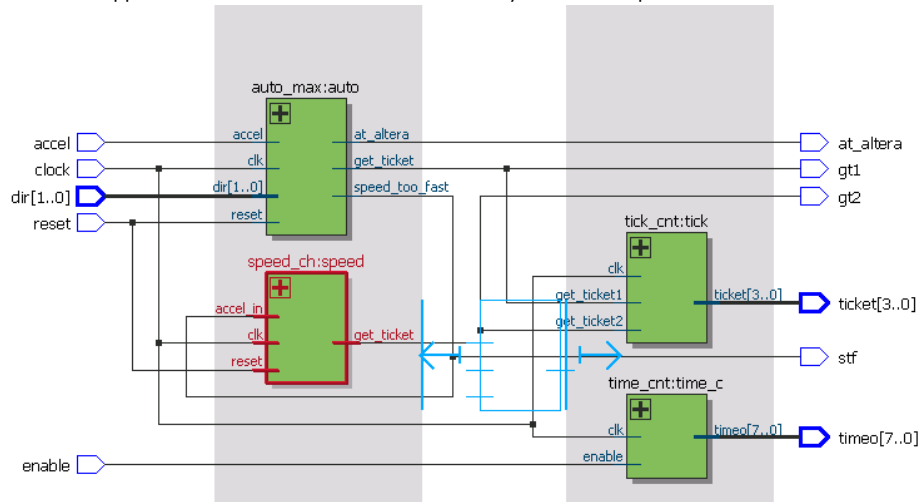
You can double-click objects in the **Connectivity Details** window to navigate to them quickly. If the plus symbol appears, you can further unwrap objects in the view. This can be very useful when tracing a signal in a complex netlist.

### 17.6.7 Moving Nodes in the Schematic View

You can drag and drop items in the schematic view to rearrange them.

**Figure 337. Drag and Drop Movement of Nodes**

To move a node from one area of the netlist to another, select the node and hold down the Shift key. Legal placements appear as shaded areas within the hierarchy. Click to drop the selected node.



Right-click and click **Refresh** to restore the schematic view to its default arrangement.

### 17.6.8 View LUT Representations in the Technology Map Viewer

You can view different representations of a LUT by right-clicking the selected LUT and clicking **Properties**.

You can view the LUT representations in the following three tabs in the **Properties** dialog box:

- The **Schematic** tab—the equivalent gate representations of the LUT.
- The **Truth Table** tab—the truth table representations.

#### Related Links

[Properties Pane](#) on page 1025

### 17.6.9 Zoom Controls

Use the Zoom Tool in the toolbar, or mouse gestures, to control the magnification of your schematic on the View menu.

By default, the Netlist Viewer displays most pages sized to fit in the window. If the schematic page is very large, the schematic is displayed at the minimum zoom level, and the view is centered on the first node. Click **Zoom In** to view the image at a larger size, and click **Zoom Out** to view the image (when the entire image is not



displayed) at a smaller size. The **Zoom** command allows you to specify a magnification percentage (100% is considered the normal size for the schematic symbols).

You can use the Zoom Tool on the Netlist Viewer toolbar to control magnification in the schematic view. When you select the Zoom Tool in the toolbar, clicking in the schematic zooms in and centers the view on the location you clicked. Right-click in the schematic to zoom out and center the view on the location you clicked. When you select the Zoom Tool, you can also zoom into a certain portion of the schematic by selecting a rectangular box area with your mouse cursor. The schematic is enlarged to show the selected area.

Within the schematic view, you can also use the following mouse gestures to zoom in on a specific section:

- **zoom in**—Dragging a box around an area starting in the upper-left and dragging to the lower right zooms in on that area.
- **zoom -0.5**—Dragging a line from lower-left to upper-right zooms out 0.5 levels of magnification.
- **zoom 0.5**—Dragging a line from lower-right to upper-left zooms in 0.5 levels of magnification.
- **zoom fit**—Dragging a line from upper-right to lower-left fits the schematic view in the page.

#### Related Links

[Filtering in the Schematic View](#) on page 1031

### 17.6.10 Navigating with the Bird's Eye View

To open the Bird's Eye View, on the View menu, click **Bird's Eye View**, or click the **Bird's Eye View** icon in the toolbar.

Viewing the entire schematic can be useful when debugging and tracing through a large netlist. The Intel Quartus Prime software allows you to quickly navigate to a specific section of the schematic using the Bird's Eye View feature, which is available in the RTL Viewer and Technology Map Viewer.

The Bird's Eye View shows the current area of interest:

- Select an area by clicking and dragging the indicator or right-clicking to form a rectangular box around an area.
- Click and drag the rectangular box to move around the schematic.
- Resize the rectangular box to zoom-in or zoom-out in the schematic view.

### 17.6.11 Partition the Schematic into Pages

For large design hierarchies, the RTL Viewer and Technology Map Viewer partition your netlist into multiple pages in the schematic view.

When a hierarchy level is partitioned into multiple pages, the title bar for the schematic window indicates which page is displayed and how many total pages exist for this level of hierarchy. The schematic view displays this as **Page** <current page number> **of** <total number of pages>.



### Related Links

[Introduction to the User Interface](#) on page 1022

## 17.6.12 Follow Nets Across Schematic Pages

Input and output connector symbols indicate nodes that connect across pages of the same hierarchy. Double-click a connector to trace the net to the next page of the hierarchy.

*Note:* After you double-click to follow a connector port, the Netlist Viewer opens a new page, which centers the view on the particular source or destination net using the same zoom factor as the previous page. To trace a specific net to the new page of the hierarchy, Intel recommends that you first select the necessary net, which highlights it in red, before you double-click to navigate across pages.

### Related Links

[Schematic Symbols](#) on page 1028

## 17.7 Cross-Probing to a Source Design File and Other Intel Quartus Prime Windows

The RTL Viewer and Technology Map Viewer allow you to cross-probe to the source design file and to various other windows in the Intel Quartus Prime software.

You can select one or more hierarchy boxes, nodes, state nodes, or state transition arcs that interest you in the Netlist Viewer and locate the corresponding items in another applicable Intel Quartus Prime software window. You can then view and make changes or assignments in the appropriate editor or floorplan.

To locate an item from the Netlist Viewer in another window, right-click the items of interest in the schematic or state diagram, point to **Locate**, and click the appropriate command. The following commands are available:

- **Locate in Assignment Editor**
- **Locate in Pin Planner**
- **Locate in Chip Planner**
- **Locate in Resource Property Editor**
- **Locate in Technology Map Viewer**
- **Locate in RTL Viewer**
- **Locate in Design File**

The options available for locating an item depend on the type of node and whether it exists after placement and routing. If a command is enabled in the menu, it is available for the selected node. You can use the **Locate in Assignment Editor** command for all nodes, but assignments might be ignored during placement and routing if they are applied to nodes that do not exist after synthesis.

The Netlist Viewer automatically opens another window for the appropriate editor or floorplan and highlights the selected node or net in the newly opened window. You can switch back to the Netlist Viewer by selecting it in the Window menu or by closing, minimizing, or moving the new window.



## 17.8 Cross-Probing to the Netlist Viewers from Other Intel Quartus Prime Windows

You can cross-probe to the RTL Viewer and Technology Map Viewer from other windows in the Intel Quartus Prime software. You can select one or more nodes or nets in another window and locate them in one of the Netlist Viewers.

You can locate nodes between the RTL Viewer and Technology Map Viewer, and you can locate nodes in the RTL Viewer and Technology Map Viewer from the following Intel Quartus Prime software windows:

- Project Navigator
- Timing Closure Floorplan
- Chip Planner
- Resource Property Editor
- Node Finder
- Assignment Editor
- Messages Window
- Compilation Report
- Timing Analyzer (supports the Technology Map Viewer only)

To locate elements in the Netlist Viewer from another Intel Quartus Prime window, select the node or nodes in the appropriate window; for example, select an entity in the **Entity** list on the **Hierarchy** tab in the Project Navigator, or select nodes in the Timing Closure Floorplan, or select node names in the **From** or **To** column in the Assignment Editor. Next, right-click the selected object, point to **Locate**, and click **Locate in RTL Viewer** or **Locate in Technology Map Viewer**. After you click this command, the Netlist Viewer opens, or is brought to the foreground if the Netlist Viewer is open.

*Note:* The first time the window opens after a compilation, the preprocessor stage runs before the Netlist Viewer opens.

The Netlist Viewer shows the selected nodes and, if applicable, the connections between the nodes. The display is similar to what you see if you right-click the object, then click **Filter > Selected Nodes** using **Filter across hierarchy**. If the nodes cannot be found in the Netlist Viewer, a message box displays the message: **Can't find requested location**.

## 17.9 Viewing a Timing Path

You can cross-probe from a report panel in the Timing Analyzer to see a visual representation of a timing path.

To take advantage of this feature, you must complete a full compilation of your design, including the timing analyzer stage. To see the timing results for your design, on the Processing menu, click **Compilation Report**. On the left side of the Compilation Report, select **Timing Analyzer**. When you select a detailed report, the timing information is listed in a table format on the right side of the Compilation Report; each row of the table represents a timing path in the design. You can also view timing paths



in Timing Analyzer report panels. To view a particular timing path in the Technology Map Viewer or RTL Viewer, right-click the appropriate row in the table, point to **Locate**, and click **Locate in Technology Map Viewer** or **Locate in RTL Viewer**.

- To locate a path, on the **Tasks** pane click **Custom Reports > Report Timing**.
- In the **Report Timing** dialog box, make necessary settings, and then click the **Report Timing** button.
- After the Timing Analyzer generates the report, right-click the node in the table and select **Locate Path**. In the Technology Map Viewer, the schematic page displays the nodes along the timing path with a summary of the total delay.

When you locate the timing path from the Timing Analyzer to the Technology Map Viewer, the interconnect and cell delay associated with each node is displayed on top of the schematic symbols. The total slack of the selected timing path is displayed in the Page Title section of the schematic.

In the RTL Viewer, the schematic page displays the nodes in the paths between the source and destination registers with a summary of the total delay.

The RTL Viewer netlist is based on an initial stage of synthesis, so the post-fitting nodes might not exist in the RTL Viewer netlist. Therefore, the internal delay numbers are not displayed in the RTL Viewer as they are in the Technology Map Viewer, and the timing path might not be displayed exactly as it appears in the timing analysis report. If multiple paths exist between the source and destination registers, the RTL Viewer might display more than just the timing path. There are also some cases in which the path cannot be displayed, such as paths through state machines, encrypted intellectual property (IP), or registers that are created during the fitting process. In cases where the timing path displayed in the RTL Viewer might not be the correct path, the compiler issues messages.

## 17.10 Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>• Implemented Intel rebranding.</li> </ul>
2016.05.03	16.0.0	Removed Schematic Viewer topic.
2015.11.02	15.1.0	Added Schematic Viewer topic for viewing stage snapshots. Added information for the following new features and feature updates: <ul style="list-style-type: none"> <li>• Nets visible across hierarchies</li> <li>• Connection Details</li> <li>• Display Settings</li> <li>• Hand Tool</li> <li>• Area Selection Tool</li> <li>• New default behavior for Show/Hide Instance Pins (default is now off)</li> </ul>
2014.06.30	14.0.0	Added Show Netlist on One Page and show/Hide Instance Pins commands.
November 2013	13.1.0	Removed HardCopy device information. Reorganized and migrated to new template. Added support for new Netlist viewer.
November 2012	12.1.0	Added sections to support Global Net Routing feature.
<i>continued...</i>		



Date	Version	Changes
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template.
July 2010	10.0.0	<ul style="list-style-type: none"> <li>Updated screenshots</li> <li>Updated chapter for the Intel Quartus Prime software version 10.0, including major user interface changes</li> </ul>
November 2009	9.1.0	<ul style="list-style-type: none"> <li>Updated devices</li> <li>Minor text edits</li> </ul>
March 2009	9.0.0	<ul style="list-style-type: none"> <li>Chapter 13 was formerly Chapter 12 in version 8.1.0</li> <li>Updated Figure 13–2, Figure 13–3, Figure 13–4, Figure 13–14, and Figure 13–30</li> <li>Added “Enable or Disable the Auto Hierarchy List” on page 13–15</li> <li>Updated “Find Command” on page 13–44</li> </ul>
November 2008	8.1.0	Changed page size to 8.5” × 11”
May 2008	8.0.0	<ul style="list-style-type: none"> <li>Added Arria GX support</li> <li>Updated operator symbols</li> <li>Updated information about the radial menu feature</li> <li>Updated zooming feature</li> <li>Updated information about probing from schematic to Signal Tap Analyzer</li> <li>Updated constant signal information</li> <li>Added .png and .gif to the list of supported image file formats</li> <li>Updated several figures and tables</li> <li>Added new sections “Enabling and Disabling the Radial Menu”, “Changing the Time Interval”, “Changing the Constant Signal Value Formatting”, “Logic Clouds in the RTL Viewer”, “Logic Clouds in the Technology Map Viewer”, “Manually Group and Ungroup Logic Clouds”, “Customizing the Shortcut Commands”</li> <li>Renamed several sections</li> <li>Removed section “Customizing the Radial Menu”</li> <li>Moved section “Grouping Combinational Logic into Logic Clouds”</li> <li>Updated document content based on the Intel Quartus Prime software version 8.0</li> </ul>

### Related Links

#### [Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 18 Mentor Graphics Precision Synthesis Support

---

### 18.1 About Precision RTL Synthesis Support

This manual delineates the support for the Mentor Graphics® Precision RTL Synthesis and Precision RTL Plus Synthesis software in the Intel Quartus Prime software, as well as key design flows, methodologies and techniques for improving your results for Intel devices. This manual assumes that you have set up, licensed, and installed the Precision Synthesis software and the Intel Quartus Prime software.

To obtain and license the Precision Synthesis software, refer to the Mentor Graphics website. To install and run the Precision Synthesis software and to set up your work environment, refer to the *Precision Synthesis Installation Guide* in the Precision Manuals Bookcase. To access the Manuals Bookcase in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

#### Related Links

[Mentor Graphics website](#)

### 18.2 Design Flow

The following steps describe a basic Intel Quartus Prime design flow using the Precision Synthesis software:

1. Create Verilog HDL or VHDL design files.
2. Create a project in the Precision Synthesis software that contains the HDL files for your design, select your target device, and set global constraints.
3. Compile the project in the Precision Synthesis software.
4. Add specific timing constraints, optimization attributes, and compiler directives to optimize the design during synthesis. With the design analysis and cross-probing capabilities of the Precision Synthesis software, you can identify and improve circuit area and performance issues using prelayout timing estimates.

*Note:* For best results, Mentor Graphics recommends specifying constraints that are as close as possible to actual operating requirements. Properly setting clock and I/O constraints, assigning clock domains, and indicating false and multicycle paths guide the synthesis algorithms more accurately toward a suitable solution in the shortest synthesis time.

5. Synthesize the project in the Precision Synthesis software.
6. Create a Intel Quartus Prime project and import the following files generated by the Precision Synthesis software into the Intel Quartus Prime project:





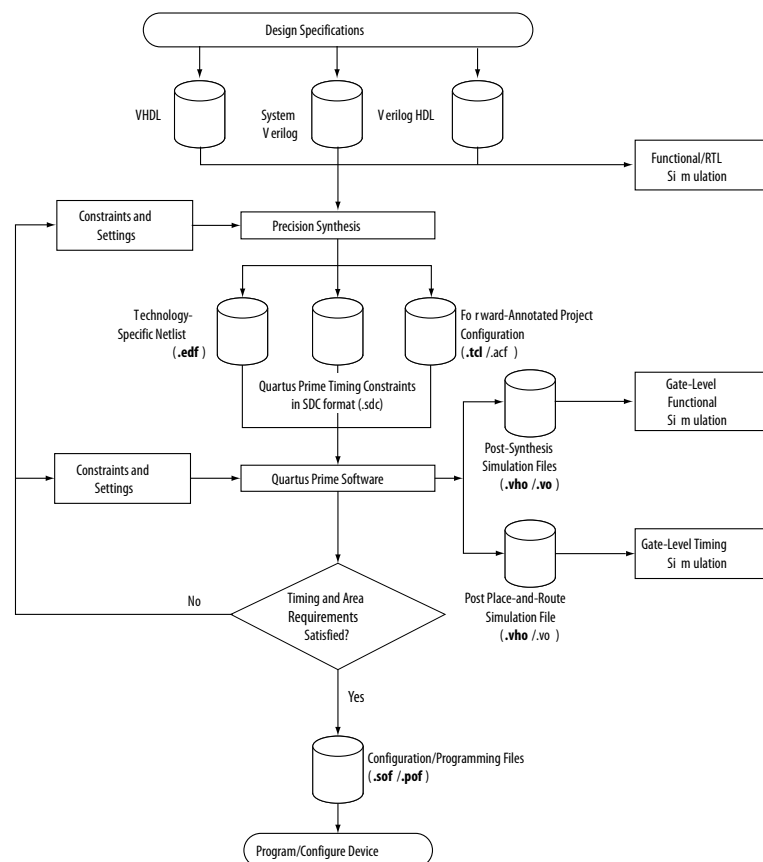
- The Verilog Quartus Mapping File ( .vqm) netlist
- Synopsys Design Constraints File ( .sdc) for Timing Analyzer constraints
- Tcl Script Files ( .tcl) to set up your Intel Quartus Prime project and pass constraints

*Note:* If your design uses the Classic Timing Analyzer for timing analysis in the Intel Quartus Prime software versions 10.0 and earlier, the Precision Synthesis software generates timing constraints in the Tcl Constraints File ( .tcl). If you are using the Intel Quartus Prime software versions 10.1 and later, you must use the Timing Analyzer for timing analysis.

7. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

You can run the Intel Quartus Prime software from within the Precision Synthesis software, or run the Precision Synthesis software using the Intel Quartus Prime software.

**Figure 338. Design Flow Using the Precision Synthesis Software and Intel Quartus Prime Software**





### 18.2.1 Timing Optimization

If your area or timing requirements are not met, you can change the constraints and resynthesize the design in the Precision Synthesis software, or you can change the constraints to optimize the design during place-and-route in the Intel Quartus Prime software. Repeat the process until the area and timing requirements are met.

You can use other options and techniques in the Intel Quartus Prime software to meet area and timing requirements. For example, the **WYSIWYG Primitive Resynthesis** option can perform optimizations on your EDIF netlist in the Intel Quartus Prime software.

While simulation and analysis can be performed at various points in the design process, final timing analysis should be performed after placement and routing is complete.

#### Related Links

- [Netlist Optimizations and Physical Synthesis documentation](#)
- [Timing Closure and Optimization documentation](#)

## 18.3 Intel Device Family Support

The Precision Synthesis software supports active devices available in the current version of the Intel Quartus Prime software. Support for newly released device families may require an overlay. Contact Mentor Graphics for more information.

## 18.4 Precision Synthesis Generated Files

During synthesis, the Precision Synthesis software produces several intermediate and output files.

**Table 276. Precision Synthesis Software Intermediate and Output Files**

File Extension	File Description
.psp	Precision Synthesis Project File.
.xdb	Mentor Graphics Design Database File.
.rep <sup>(14)</sup>	Synthesis Area and Timing Report File.
.vqm <sup>(15)</sup>	Technology-specific netlist in .vqm file format.

*continued...*

<sup>(14)</sup> The timing report file includes performance estimates that are based on pre-place-and-route information. Use the  $f_{MAX}$  reported by the Intel Quartus Prime software after place-and-route for accurate post-place-and-route timing information. The area report file includes post-synthesis device resource utilization statistics that can differ from the resource usage after place-and-route due to black boxes or further optimizations performed during placement and routing. Use the device utilization reported by the Intel Quartus Prime software after place-and-route for final resource utilization results.

<sup>(15)</sup> The Precision Synthesis software-generated VQM file is supported by the Intel Quartus Prime software version 10.1 and later.



File Extension	File Description
	By default, the Precision Synthesis software creates .vqm files for Arria series, Cyclone series, and Stratix series devices. The Precision Synthesis software defaults to creating .vqm files when the device is supported.
.tcl	Forward-annotated Tcl assignments and constraints file. The <project name>.tcl file is generated for all devices. The .tcl file acts as the Intel Quartus Prime Project Configuration file and is used to make basic project and placement assignments, and to create and compile a Intel Quartus Prime project.
.acf	Assignment and Configurations file for backward compatibility with the MAX+PLUS II software. For devices supported by the MAX+PLUS II software, the MAX+PLUS II assignments are imported from the MAX+PLUS II .acf file.
.sdc	Intel Quartus Prime timing constraints file in Synopsys Design Constraints format. This file is generated automatically if the device uses the Timing Analyzer by default in the Intel Quartus Prime software, and has the naming convention <project name>_pnr_constraints.sdc.

### Related Links

[Synthesizing the Design and Evaluating the Results](#) on page 1046

## 18.5 Creating and Compiling a Project in the Precision Synthesis Software

After creating your design files, create a project in the Precision Synthesis software that contains the basic settings for compiling the design.

## 18.6 Mapping the Precision Synthesis Design

In the next steps, you set constraints and map the design to technology-specific cells. The Precision Synthesis software maps the design by default to the fastest possible implementation that meets your timing constraints. To accomplish this, you must specify timing requirements for the automatically determined clock sources. With this information, the Precision Synthesis software performs static timing analysis to determine the location of the critical timing paths. The Precision Synthesis software achieves the best results for your design when you set as many realistic constraints as possible. Be sure to set constraints for timing, mapping, false paths, multicycle paths, and other factors that control the structure of the implemented design.

Mentor Graphics recommends creating an **.sdc** file and adding this file to the **Constraint Files** section of the **Project Files** list. You can create this file with a text editor, by issuing command-line constraint parameters, or by directing the Precision Synthesis software to generate the file automatically the first time you synthesize your design. By default, the Precision Synthesis software saves all timing constraints and attributes in two files: **precision\_rtl.sdc** and **precision\_tech.sdc**. The **precision\_rtl.sdc** file contains constraints set on the RTL-level database (post-compilation) and the **precision\_tech.sdc** file contains constraints set on the gate-level database (post-synthesis) located in the current implementation directory.

You can also enter constraints at the command line. After adding constraints at the command line, update the **.sdc** file with the `update constraint file` command. You can add constraints that change infrequently directly to the HDL source files with HDL attributes or pragmas.



**Note:** The Precision **.sdc** file contains all the constraints for the Precision Synthesis project. For the Intel Quartus Prime software, placement constraints are written in a **.tcl** file and timing constraints for the Timing Analyzer are written in the Intel Quartus Prime **.sdc** file.

For details about the syntax of Synopsys Design Constraint commands, refer to the *Precision RTL Synthesis User's Manual* and the *Precision Synthesis Reference Manual*. For more details and examples of attributes, refer to the *Attributes* chapter in the *Precision Synthesis Reference Manual*.

### 18.6.1 Setting Timing Constraints

The Precision Synthesis software uses timing constraints, based on the industry-standard **.sdc** file format, to deliver optimal results. Missing timing constraints can result in incomplete timing analysis and might prevent timing errors from being detected. The Precision Synthesis software provides constraint analysis prior to synthesis to ensure that designs are fully and accurately constrained. The `<project name>_pnr_constraints.sdc` file, which contains timing constraints in SDC format, is generated in the Intel Quartus Prime software.

**Note:** Because the **.sdc** file format requires that timing constraints be set relative to defined clocks, you must specify your clock constraints before applying any other timing constraints.

You also can use multicycle path and false path assignments to relax requirements or exclude nodes from timing requirements, which can improve area utilization and allow the software optimizations to focus on the most critical parts of the design.

For details about the syntax of Synopsys Design Constraint commands, refer to the *Precision RTL Synthesis User's Manual* and the *Precision Synthesis Reference Manual*.

### 18.6.2 Setting Mapping Constraints

Mapping constraints affect how your design is mapped into the target Intel device. You can set mapping constraints in the user interface, in HDL code, or with the `set_attribute` command in the constraint file.

### 18.6.3 Assigning Pin Numbers and I/O Settings

The Precision Synthesis software supports assigning device pin numbers, I/O standards, drive strengths, and slew rate settings to top-level ports of the design. You can set these timing constraints with the `set_attribute` command, the GUI, or by specifying synthesis attributes in your HDL code. These constraints are forward-annotated in the `<project name>.tcl` file that is read by the Intel Quartus Prime software during place-and-route and do not affect synthesis.

You can use the `set_attribute` command in the Precision Synthesis software **.sdc** file to specify pin number constraints, I/O standards, drive strengths, and slow slew-rate settings. The table below describes the format to use for entries in the Precision Synthesis software constraint file.



**Table 277. Constraint File Settings**

Constraint	Entry Format for Precision Constraint File
Pin number	<code>set_attribute -name PIN_NUMBER -value "&lt;pin number&gt;" -port &lt;port name&gt;</code>
I/O standard	<code>set_attribute -name IOSTANDARD -value "&lt;I/O Standard&gt;" -port &lt;port name&gt;</code>
Drive strength	<code>set_attribute -name DRIVE -value "&lt;drive strength in mA&gt;" -port &lt;port name&gt;</code>
Slew rate	<code>set_attribute -name SLEW -value "TRUE   FALSE" -port &lt;port name&gt;</code>

You also can use synthesis attributes or pragmas in your HDL code to make these assignments.

**Example 105. Verilog HDL Pin Assignment**

```
//pragma attribute clk pin_number P10;
```

**Example 106. VHDL Pin Assignment**

```
attribute pin_number : string
attribute pin_number of clk : signal is "P10";
```

You can use the same syntax to assign the I/O standard using the `IOSTANDARD` attribute, drive strength using the attribute `DRIVE`, and slew rate using the `SLEW` attribute.

For more details about attributes and how to set these attributes in your HDL code, refer to the *Precision Synthesis Reference Manual*.

### 18.6.4 Assigning I/O Registers

The Precision Synthesis software performs timing-driven I/O register mapping by default. You can force a register to the device IO element (IOE) using the Complex I/O constraint. This option does not apply if you turn off **I/O pad insertion**.

*Note:* You also can make the assignment by right-clicking on the pin in the Schematic Viewer.

For the Stratix series, Cyclone series, and the MAX II device families, the Precision Synthesis software can move an internal register to an I/O register without any restrictions on design hierarchy.

For more mature devices, the Precision Synthesis software can move an internal register to an I/O register only when the register exists in the top-level of the hierarchy. If the register is buried in the hierarchy, you must flatten the hierarchy so that the buried registers are moved to the top-level of the design.

### 18.6.5 Disabling I/O Pad Insertion

The Precision Synthesis software assigns I/O pad atoms (device primitives used to represent the I/O pins and I/O registers) to all ports in the top-level of a design by default. In certain situations, you might not want the software to add I/O pads to all



I/O pins in the design. The Intel Quartus Prime software can compile a design without I/O pads; however, including I/O pads provides the Precision Synthesis software with more information about the top-level pins in the design.

### 18.6.5.1 Preventing the Precision Synthesis Software from Adding I/O Pads

If you are compiling a subdesign as a separate project, I/O pins cannot be primary inputs or outputs of the device; therefore, the I/O pins should not have an I/O pad associated with them.

To prevent the Precision Synthesis software from adding I/O pads:

- You can use the Precision Synthesis GUI or add the following command to the project file:

```
setup_design -addio=false
```

### 18.6.5.2 Preventing the Precision Synthesis Software from Adding an I/O Pad on an Individual Pin

To prevent I/O pad insertion on an individual pin when you are using a black box, such as DDR or a phase-locked loop (PLL), at the external ports of the design, perform the following steps:

1. Compile your design.
2. Use the Precision Synthesis GUI to select the individual pin and turn off I/O pad insertion.

**Note:** You also can make this assignment by attaching the `nopad` attribute to the port in the HDL source code.

## 18.6.6 Controlling Fan-Out on Data Nets

Fan-out is defined as the number of nodes driven by an instance or top-level port. High fan-out nets can cause significant delays that result in an unroutable net. On a critical path, high fan-out nets can cause longer delays in a single net segment that result in the timing constraints not being met. To prevent this behavior, each device family has a global fan-out value set in the Precision Synthesis software library. In addition, the Intel Quartus Prime software automatically routes high fan-out signals on global routing lines in the Intel device whenever possible.

To eliminate routability and timing issues associated with high fan-out nets, the Precision Synthesis software also allows you to override the library default value on a global or individual net basis. You can override the library value by setting a `max_fanout` attribute on the net.

## 18.7 Synthesizing the Design and Evaluating the Results

During synthesis, the Precision Synthesis software optimizes the compiled design, and then writes out netlists and reports to the implementation subdirectory of your working directory after the implementation is saved, using the following naming convention:

```
<project name>_impl_<number>
```



After synthesis is complete, you can evaluate the results for area and timing. The *Precision RTL Synthesis User's Manual* describes different results that can be evaluated in the software.

There are several schematic viewers available in the Precision Synthesis software: RTL schematic, Technology-mapped schematic, and Critical Path schematic. These analysis tools allow you to quickly and easily isolate the source of timing or area issues, and to make additional constraint or code changes to optimize the design.

### 18.7.1 Obtaining Accurate Logic Utilization and Timing Analysis Reports

Historically, designers have relied on post-synthesis logic utilization and timing reports to determine the amount of logic their design requires, the size of the device required, and how fast the design runs. However, today's FPGA devices provide a wide variety of advanced features in addition to basic registers and look-up tables (LUTs). The Intel Quartus Prime software has advanced algorithms to take advantage of these features, as well as optimization techniques to increase performance and reduce the amount of logic required for a given design. In addition, designs can contain black boxes and functions that take advantage of specific device features. Because of these advances, synthesis tool reports provide post-synthesis area and timing estimates, but you should use the place-and-route software to obtain final logic utilization and timing reports.

## 18.8 Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including the LPMs, and device-specific Intel FPGA IP, and IP available through third-party partners. You can use IP cores by instantiating them in your HDL code or by inferring certain functions from generic HDL code.

If you want to instantiate an IP core such as a PLL in your HDL code, you can instantiate and parameterize the function using the port and parameter definitions, or you can customize a function with the parameter editor. Intel recommends using the IP Catalog and parameter editor, which provides a graphical interface within the Intel Quartus Prime software for customizing and parameterizing any available IP core for the design.

The Precision Synthesis software automatically recognizes certain types of HDL code and infers the appropriate IP core.

#### Related Links

- [Inferring Intel FPGA IP Cores from HDL Code](#) on page 1050
- [Recommended HDL Coding Styles documentation](#) on page 100
- [Introduction to Intel FPGA IP Cores documentation](#)

### 18.8.1 Instantiating IP Cores With IP Catalog-Generated Verilog HDL Files

The IP Catalog generates a Verilog HDL instantiation template file `<output file>_inst.v` and a hollow-body black box module declaration `<output file>_bb.v` for use in your Precision Synthesis design. Incorporate the instantiation template file, `<output file>_inst.v`, into your top-level design to instantiate the IP core wrapper file, `<output file>.v`.



Include the hollow-body black box module declaration `<output file>_bb.v` in your Precision Synthesis project to describe the port connections of the black box. Adding the IP core wrapper file `<output file>.v` in your Precision Synthesis project is optional, but you must add it to your Intel Quartus Prime project along with the Precision Synthesis-generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file `<output file>.v` in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Intel Quartus Prime software during place-and-route.

### 18.8.2 Instantiating IP Cores With IP Catalog-Generated VHDL Files

The IP Catalog generates a VHDL component declaration file `<output file>.cmp` and a VHDL instantiation template file `<output file>_inst.vhd` for use in your Precision Synthesis design. Incorporate the component declaration and instantiation template into your top-level design to instantiate the IP core wrapper file, `<output file>.vhd`.

Adding the IP core wrapper file `<output file>.vhd` in your Precision Synthesis project is optional, but you must add the file to your Intel Quartus Prime project along with the Precision Synthesis-generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file `<output file>.v` in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Intel Quartus Prime software during place-and-route.

### 18.8.3 Instantiating Intellectual Property With the IP Catalog and Parameter Editor

Many Intel FPGA IP functions include a resource and timing estimation netlist that the Precision Synthesis software can use to synthesize and optimize logic around the IP efficiently. As a result, the Precision Synthesis software provides better timing correlation, area estimates, and Quality of Results (QoR) than a black box approach.

To create this netlist file, perform the following steps:

1. Select the IP function in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Intel Quartus Prime software generates a file `<output file>_syn.v`. This netlist contains the “grey box” information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file into your Precision Synthesis project as an input file. Then include the IP core wrapper file `<output file>.v|vhd` in the Intel Quartus Prime project along with your EDIF or VQM output netlist.





The generated “grey box” netlist file, *<output file>\_syn.v*, is always in Verilog HDL format, even if you select VHDL as the output file format.

**Note:** For information about creating a grey box netlist file from the command line, search Altera's Knowledge Database.

#### Related Links

[Altera Knowledge Center website](#)

### 18.8.4 Instantiating Black Box IP Functions With Generated Verilog HDL Files

You can use the `syn_black_box` or `black_box` compiler directives to declare a module as a black box. The top-level design files must contain the IP port mapping and a hollow-body module declaration. You can apply the directive to the module declaration in the top-level file or a separate file included in the project so that the Precision Synthesis software recognizes the module is a black box.

**Note:** The `syn_black_box` and `black_box` directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates **my\_verilogIP.v**, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

#### Example 107.

#### Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
    input clk;
    output[7:0] count;

    my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule

// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
    input clock;
    output[7:0] q;
endmodule
```

### 18.8.5 Instantiating Black Box IP Functions With Generated VHDL Files

You can use the `syn_black_box` or `black_box` compiler directives to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port mapping. Apply the directive to the component declaration in the top-level file.

**Note:** The `syn_black_box` and `black_box` directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates **my\_vhdlIP.vhd**, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

**Example 108. Top-Level VHDL Code with Black Box Instantiation of IP**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY top IS
  PORT (
    clk: IN STD_LOGIC ;
    count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END top;

ARCHITECTURE rtl OF top IS
  COMPONENT my_vhdlIP
  PORT (
    clock: IN STD_LOGIC ;
    q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
  end COMPONENT;
  attribute syn_black_box : boolean;
  attribute syn_black_box of my_vhdlIP: component is true;
  BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
      clock => clk,
      q => count
    );
  END rtl;
```

**18.8.6 Inferring Intel FPGA IP Cores from HDL Code**

The Precision Synthesis software automatically recognizes certain types of HDL code and maps arithmetical operators, relational operators, and memory (RAM and ROM), to technology-specific implementations. This functionality allows technology-specific resources to implement these structures by inferring the appropriate Intel function to provide optimal results. In some cases, the Precision Synthesis software has options that you can use to disable or control inference.

For coding style recommendations and examples for inferring technology-specific architecture in Intel devices, refer to the *Precision Synthesis Style Guide*.

**Related Links**

[Recommended HDL Coding Styles documentation](#) on page 100

**18.8.6.1 Multipliers**

The Precision Synthesis software detects multipliers in HDL code and maps them directly to device atoms to implement the multiplier in the appropriate type of logic. The Precision Synthesis software also allows you to control the device resources that are used to implement individual multipliers.

**18.8.6.1.1 Controlling DSP Block Inference for Multipliers**

By default, the Precision Synthesis software uses DSP blocks available in Stratix series devices to implement multipliers. The default setting is **AUTO**, which allows the Precision Synthesis software to map to logic look-up tables (LUTs) or DSP blocks, depending on the size of the multiplier. You can use the Precision Synthesis GUI or HDL attributes for direct mapping to only logic elements or to only DSP blocks.



**Table 278. Options for dedicated\_mult Parameter to Control Multiplier Implementation in Precision Synthesis**

Value	Description
<b>ON</b>	Use only DSP blocks to implement multipliers, regardless of the size of the multiplier.
<b>OFF</b>	Use only logic (LUTs) to implement multipliers, regardless of the size of the multiplier.
<b>AUTO</b>	Use logic (LUTs) or DSP blocks to implement multipliers, depending on the size of the multipliers.

### 18.8.6.2 Setting the Use Dedicated Multiplier Option

To set the Use Dedicated Multiplier option in the Precision Synthesis GUI, compile the design, and then in the Design Hierarchy browser, right-click the operator for the desired multiplier and click **Use Dedicated Multiplier**.

### 18.8.6.3 Setting the dedicated\_mult Attribute

To control the implementation of a multiplier in your HDL code, use the `dedicated_mult` attribute with the appropriate value as shown in the examples below.

#### Example 109. Setting the dedicated\_mult Attribute in Verilog HDL

```
//synthesis attribute <signal name> dedicated_mult <value>
```

#### Example 110. Setting the dedicated\_mult Attribute in VHDL

```
ATTRIBUTE dedicated_mult: STRING;  
ATTRIBUTE dedicated_mult OF <signal name>: SIGNAL IS <value>;
```

The `dedicated_mult` attribute can be applied to signals and wires; it does not work when applied to a register. This attribute can be applied only to simple multiplier code, such as `a = b * c`.

Some signals for which the `dedicated_mult` attribute is set can be removed during synthesis by the Precision Synthesis software for design optimization. In such cases, if you want to force the implementation, you should preserve the signal by setting the `preserve_signal` attribute to `TRUE`.

#### Example 111. Setting the preserve\_signal Attribute in Verilog HDL

```
//synthesis attribute <signal name> preserve_signal TRUE
```

#### Example 112. Setting the preserve\_signal Attribute in VHDL

```
ATTRIBUTE preserve_signal: BOOLEAN;  
ATTRIBUTE preserve_signal OF <signal name>: SIGNAL IS TRUE;
```

**Example 113. Verilog HDL Multiplier Implemented in Logic**

```
module unsigned_mult (result, a, b);
    output [15:0] result;
    input [7:0] a;
    input [7:0] b;
    assign result = a * b;
    //synthesis attribute result dedicated_mult OFF
endmodule
```

**Example 114. VHDL Multiplier Implemented in Logic**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY unsigned_mult IS
    PORT(
        a: IN std_logic_vector (7 DOWNTO 0);
        b: IN std_logic_vector (7 DOWNTO 0);
        result: OUT std_logic_vector (15 DOWNTO 0));
    ATTRIBUTE dedicated_mult: STRING;
END unsigned_mult;

ARCHITECTURE rtl OF unsigned_mult IS
    SIGNAL a_int, b_int: UNSIGNED (7 downto 0);
    SIGNAL pdt_int: UNSIGNED (15 downto 0);
    ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF";
BEGIN
    a_int <= UNSIGNED (a);
    b_int <= UNSIGNED (b);
    pdt_int <= a_int * b_int;
    result <= std_logic_vector(pdt_int);
END rtl;
```

**18.8.6.4 Multiplier-Accumulators and Multiplier-Adders**

The Precision Synthesis software also allows you to control the device resources used to implement multiply-accumulators or multiply-adders in your project or in a particular module.

The Precision Synthesis software detects multiply-accumulators or multiply-adders in HDL code and infers an ALTMULT\_ACCUM or ALTMULT\_ADD IP cores so that the logic can be placed in DSP blocks, or the software maps these functions directly to device atoms to implement the multiplier in the appropriate type of logic.

**Note:** The Precision Synthesis software supports inference for these functions only if the target device family has dedicated DSP blocks.

For more information about DSP blocks in Intel devices, refer to the appropriate Intel device family handbook and device-specific documentation. For details about which functions a given DSP block can implement, refer to the DSP Solutions Center on the Altera website.

For more information about inferring multiply-accumulator and multiply-adder IP cores in HDL code, refer to the Intel *Recommended HDL Coding Styles* and the Mentor Graphics *Precision Synthesis Style Guide*.



### Related Links

- [Altera DSP Solutions website](#)
- [Recommended HDL Coding Styles documentation](#) on page 100

### 18.8.6.5 Controlling DSP Block Inference

By default, the Precision Synthesis software infers the ALTMULT\_ADD or ALTMULT\_ACCUM IP cores appropriately in your design. These IP cores allow the Intel Quartus Prime software to select either logic or DSP blocks, depending on the device utilization and the size of the function.

You can use the `extract_mac` attribute to prevent inference of an ALTMULT\_ADD or ALTMULT\_ACCUM IP cores in a certain module or entity.

**Table 279. Options for `extract_mac` Attribute Controlling DSP Implementation**

Value	Description
TRUE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is inferred.
FALSE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is not inferred.

To control inference, use the `extract_mac` attribute with the appropriate value from the examples below in your HDL code.

#### Example 115. Setting the `extract_mac` Attribute in Verilog HDL

```
//synthesis attribute <module name> extract_mac <value>
```

#### Example 116. Setting the `extract_mac` Attribute in VHDL

```
ATTRIBUTE extract_mac: BOOLEAN;  
ATTRIBUTE extract_mac OF <entity name>: ENTITY IS <value>;
```

To control the implementation of the multiplier portion of a multiply-accumulator or multiply-adder, you must use the `dedicated_mult` attribute.

You can use the `extract_mac`, `dedicated_mult`, and `preserve_signal` attributes (in Verilog HDL and VHDL) to implement the given DSP function in logic in the Intel Quartus Prime software.

#### Example 117. Using `extract_mac`, `dedicated_mult`, and `preserve_signal` in Verilog HDL

```
module unsig_altmult_accuml (dataout, dataa, datab, clk, aclr, clken);  
    input [7:0] dataa, datab;  
    input clk, aclr, clken;  
    output [31:0] dataout;  
  
    reg [31:0] dataout;  
    wire [15:0] multa;  
    wire [31:0] adder_out;  
  
    assign multa = dataa * datab;  
  
    //synthesis attribute multa preserve_signal TRUE  
    //synthesis attribute multa dedicated_mult OFF  
    assign adder_out = multa + dataout;
```

```

always @ (posedge clk or posedge aclr)
begin
  if (aclr)
    dataout <= 0;
  else if (clken)
    dataout <= adder_out;
end

//synthesis attribute unsig_altmult_accuml extract_mac FALSE
endmodule

```

### Example 118. Using extract\_mac, dedicated\_mult, and preserve\_signal in VHDL

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
ENTITY signedmult_add IS
  PORT(
    a, b, c, d: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    result: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
  ATTRIBUTE preserve_signal: BOOLEAN;
  ATTRIBUTE dedicated_mult: STRING;
  ATTRIBUTE extract_mac: BOOLEAN;
  ATTRIBUTE extract_mac OF signedmult_add: ENTITY IS FALSE;
END signedmult_add;
ARCHITECTURE rtl OF signedmult_add IS
  SIGNAL a_int, b_int, c_int, d_int : signed (7 DOWNTO 0);
  SIGNAL pdt_int, pdt2_int : signed (15 DOWNTO 0);
  SIGNAL result_int: signed (15 DOWNTO 0);
  ATTRIBUTE preserve_signal OF pdt_int: SIGNAL IS TRUE;
  ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF";
  ATTRIBUTE preserve_signal OF pdt2_int: SIGNAL IS TRUE;
  ATTRIBUTE dedicated_mult OF pdt2_int: SIGNAL IS "OFF";
BEGIN
  a_int <= signed (a);
  b_int <= signed (b);
  c_int <= signed (c);
  d_int <= signed (d);
  pdt_int <= a_int * b_int;
  pdt2_int <= c_int * d_int;
  result_int <= pdt_int + pdt2_int;
  result <= STD_LOGIC_VECTOR(result_int);
END rtl;

```

#### 18.8.6.6 RAM and ROM

The Precision Synthesis software detects memory structures in HDL code and converts them to an operator that infers an ALTSYNCRAM or LPM\_RAM\_DP IP cores, depending on the device family. The software then places these functions in memory blocks.

The software supports inference for these functions only if the target device family has dedicated memory blocks.

For more information about inferring RAM and ROM IP cores in HDL code, refer to the *Precision Synthesis Style Guide*.

#### Related Links

[Recommended HDL Coding Styles documentation](#) on page 100



## 18.9 Document Revision History

**Table 280. Document Revision History**

Date	Version	Changes
2016.10.31	16.1.0	<ul style="list-style-type: none"> <li>Implemented Intel rebranding.</li> </ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"> <li>Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li> </ul>
June 2014	14.0.0	<ul style="list-style-type: none"> <li>Dita conversion.</li> <li>Removed obsolete devices.</li> <li>Replaced Intel FPGA IP, MegaWizard, and IP Toolbench content with IP Catalog and Parameter Editor content.</li> </ul>
June 2012	12.0.0	<ul style="list-style-type: none"> <li>Removed survey link.</li> </ul>
November 2011	10.1.1	<ul style="list-style-type: none"> <li>Template update.</li> <li>Minor editorial changes.</li> </ul>
December 2010	10.1.0	<ul style="list-style-type: none"> <li>Changed to new document template.</li> <li>Removed Classic Timing Analyzer support.</li> <li>Added support for .vqm netlist files.</li> <li>Edited the "Creating Intel Quartus Prime Projects for Multiple EDIF Files" on page 15–30 section for changes with the incremental compilation flow.</li> <li>Editorial changes.</li> </ul>
July 2010	10.0.0	<ul style="list-style-type: none"> <li>Minor updates for the Intel Quartus Prime software version 10.0 release</li> </ul>
November 2009	9.1.0	<ul style="list-style-type: none"> <li>Minor updates for the Intel Quartus Prime software version 9.1 release</li> </ul>
March 2009	9.0.0	<ul style="list-style-type: none"> <li>Updated list of supported devices for the Intel Quartus Prime software version 9.0 release</li> <li>Chapter 11 was previously Chapter 10 in software version 8.1</li> </ul>
November 2008	8.1.0	<ul style="list-style-type: none"> <li>Changed to 8-1/2 x 11 page size</li> <li>Title changed to <i>Mentor Graphics Precision Synthesis Support</i></li> <li>Updated list of supported devices</li> <li>Added information about the Precision RTL Plus incremental synthesis flow</li> <li>Updated Figure 10-1 to include SystemVerilog</li> <li>Updated "Guidelines for Intel FPGA IP and Architecture-Specific Features" on page 10–19</li> <li>Updated "Incremental Compilation and Block-Based Design" on page 10–28</li> <li>Added section "Creating Partitions with the incr_partition Attribute" on page 10–29</li> </ul>
May 2008	8.0.0	<ul style="list-style-type: none"> <li>Removed Mercury from the list of supported devices</li> <li>Changed Precision version to 2007a update 3</li> <li>Added note for Stratix IV support</li> <li>Renamed "Creating a Project and Compiling the Design" section to "Creating and Compiling a Project in the Precision RTL Synthesis Software"</li> <li>Added information about constraints in the Tcl file</li> <li>Updated document based on the Intel Quartus Prime software version 8.0</li> </ul>

### Related Links

#### [Documentation Archive](#)

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.



## 19 Synopsys Synplify Support

---

### 19.1 About Synplify Support

the Intel Quartus Prime software supports use of the Synopsys Synplify software design flows, methodologies, and techniques for achieving optimal results in Intel devices. Synplify support applies to Synplify, Synplify Pro, and Synplify Premier software. This document assumes proper set up, licensing, and basic familiarity with the Synplify software.

This document covers the following information:

- General design flow with the Synplify and Intel Quartus Prime software.
- Synplify software optimization strategies, including timing-driven compilation settings, optimization options, and other attributes.
- Guidelines for use of Quartus Prime IP cores, including guidelines for HDL inference of IP cores.

#### Related Links

- [Synplify Synthesis Techniques with the Intel Quartus Prime Software online training](#)
- [Synplify Pro Tips and Tricks online training](#)

### 19.2 Design Flow

The following steps describe a basic Intel Quartus Prime software design flow using the Synplify software:

1. Create Verilog HDL (.v) or VHDL (.vhd) design files.
2. Set up a project in the Synplify software and add the HDL design files for synthesis.
3. Select a target device and add timing constraints and compiler directives in the Synplify software to help optimize the design during synthesis.
4. Synthesize the project in the Synplify software.
5. Create a Intel Quartus Prime project and import the following files generated by the Synplify software into the Intel Quartus Prime software. Use the following files for placement and routing, and for performance evaluation:



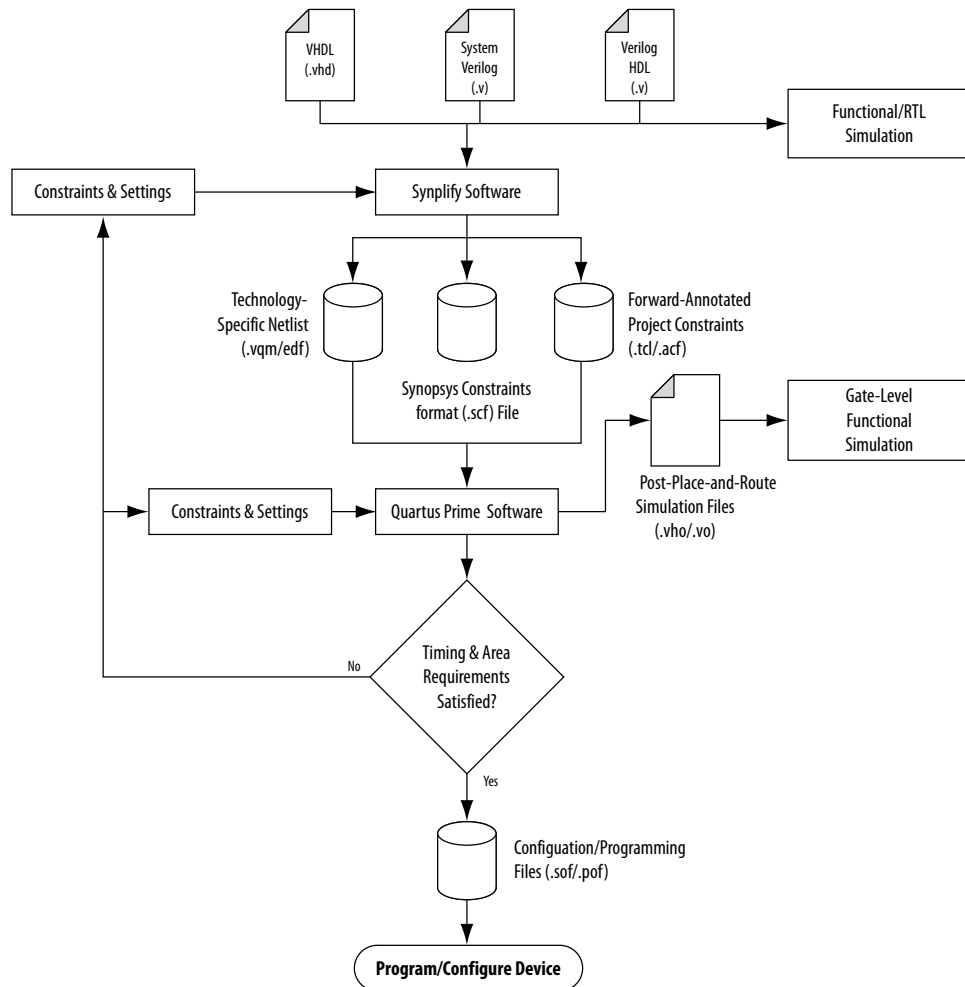


- Verilog Quartus Mapping File (**.vqm**) netlist.
- The Synopsys Constraints Format (**.scf**) file for Timing Analyzer constraints.
- The **.tcl** file to set up your Intel Quartus Prime project and pass constraints.

*Note:* Alternatively, you can run the Intel Quartus Prime software from within the Synplify software.

6. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

**Figure 339. Recommended Design Flow**



**Related Links**

- [Synplify Software Generated Files](#) on page 1058
- [Design Constraints Support](#) on page 1059



## 19.3 Hardware Description Language Support

The Synplify software supports VHDL, Verilog HDL, and SystemVerilog source files. However, only the Synplify Pro and Premier software support mixed synthesis, allowing a combination of VHDL and Verilog HDL or SystemVerilog format source files.

The HDL Analyst that is included in the Synplify software is a graphical tool for generating schematic views of the technology-independent RTL view netlist (**.srs**) and technology-view netlist (**.srm**) files. You can use the Synplify HDL Analyst to analyze and debug your design visually. The HDL Analyst supports cross-probing between the RTL and Technology views, the HDL source code, the Finite State Machine (FSM) viewer, and between the technology view and the timing report file in the Intel Quartus Prime software. A separate license file is required to enable the HDL Analyst in the Synplify software. The Synplify Pro and Premier software include the HDL Analyst.

### Related Links

[Guidelines for Intel FPGA IP Cores and Architecture-Specific Features](#) on page 1068

## 19.4 Intel Device Family Support

Support for newly released device families may require an overlay. Contact Synopsys for more information.

### Related Links

[Synopsys Website](#)

## 19.5 Tool Setup

### 19.5.1 Specifying the Intel Quartus Prime Software Version

You can specify your version of the Intel Quartus Prime software in **Implementation Options** in the Synplify software. This option ensures that the netlist is compatible with the software version and supports the newest features. Intel recommends using the latest version of the Intel Quartus Prime software whenever possible. If your Intel Quartus Prime software version is newer than the versions available in the **Quartus Version** list, check if there is a newer version of the Synplify software available that supports the current Intel Quartus Prime software version. Otherwise, select the latest version in the list for the best compatibility.

*Note:* The **Quartus Version** list is available only after selecting an Intel device.

#### Example 119. Specifying Intel Quartus Prime Software Version at the Command Line

```
set_option -quartus_version <version number>
```

## 19.6 Synplify Software Generated Files

During synthesis, the Synplify software produces several intermediate and output files.



**Table 281. Synplify Intermediate and Output Files**

File Extensions	File Description
<b>.vqm</b>	Technology-specific netlist in <b>.vqm</b> file format. A <b>.vqm</b> file is created for all Intel device families supported by the Intel Quartus Prime software.
<b>.scf</b> <sup>(16)</sup>	Synopsys Constraint Format file containing timing constraints for the Timing Analyzer.
<b>.tcl</b>	Forward-annotated constraints file containing constraints and assignments. A <b>.tcl</b> file for the Intel Quartus Prime software is created for all devices. The <b>.tcl</b> file contains the appropriate Tcl commands to create and set up a Intel Quartus Prime project and pass placement constraints.
<b>.srs</b>	Technology-independent RTL netlist file that can be read only by the Synplify software.
<b>.srm</b>	Technology view netlist file.
<b>.acf</b>	Assignment and Configurations file for backward compatibility with the MAX+PLUS II software. For devices supported by the MAX+PLUS II software, the MAX+PLUS II assignments are imported from the MAX+PLUS II <b>.acf</b> file.
<b>.srr</b> <sup>(17)</sup>	Synthesis Report file.

**Related Links**

[Design Flow](#) on page 1056

## 19.7 Design Constraints Support

You can specify timing constraints and attributes by using the SCOPE window of the Synplify software, by editing the **.sdc** file, or by defining the compiler directives in the HDL source file. The Synplify software forward-annotates many of these constraints to the Intel Quartus Prime software.

After synthesis is complete, do the following steps:

1. Import the **.vqm** netlist to the Intel Quartus Prime software for place-and-route.
2. Use the **.tcl** file generated by the Synplify software to forward-annotate your project constraints including device selection. The **.tcl** file calls the generated **.scf** to forward-annotate Timing Analyzer timing constraints.

**Related Links**

- [Design Flow](#) on page 1056

---

<sup>(16)</sup> If your design uses the Classic Timing Analyzer for timing analysis in the Intel Quartus Prime software versions 10.0 and earlier, the Synplify software generates timing constraints in the Tcl Constraints File (**.tcl**). If you are using the Intel Quartus Prime software versions 10.1 and later, you must use the Timing Analyzer for timing analysis.

<sup>(17)</sup> This report file includes performance estimates that are often based on pre-place-and-route information. Use the  $f_{MAX}$  reported by the Intel Quartus Prime software after place-and-route—it is the only reliable source of timing information. This report file includes post-synthesis device resource utilization statistics that might inaccurately predict resource usage after place-and-route. The Synplify software does not account for black box functions nor for logic usage reduction achieved through register packing performed by the Intel Quartus Prime software. Register packing combines a single register and look-up table (LUT) into a single logic cell, reducing logic cell utilization below the Synplify software estimate. Use the device utilization reported by the Intel Quartus Prime software after place-and-route.



- [Synplify Optimization Strategies](#) on page 1061
- [Netlist Optimizations and Physical Synthesis Documentation](#)

### 19.7.1 Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script

You can run the Intel Quartus Prime software with a Synplify-generated Tcl script.

To run the Tcl script to set up your project assignments, perform the following steps:

1. Ensure the **.vqm**, **.scf**, and **.tcl** files are located in the same directory.
2. In the Intel Quartus Prime software, on the View menu, point to and click **Tcl Console**. The Intel Quartus Prime Tcl Console opens.
3. At the Tcl Console command prompt, type the following:

```
source <path>/<project name>_cons.tcl
```

### 19.7.2 Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software

The Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry standard constraints format, Synopsys Design Constraints (SDC).

The Synplify-generated **.tcl** file contains constraints for the Intel Quartus Prime software, such as the device specification and any location constraints. Timing constraints are forward-annotated in the Synopsys Constraints Format (**.scf**) file.

**Note:** Synopsys recommends that you modify constraints using the SCOPE constraint editor window, rather than using the generated **.sdc**, **.scf**, or **.tcl** file.

The following list of Synplify constraints are converted to the equivalent Intel Quartus Prime SDC commands and are forward-annotated to the Intel Quartus Prime software in the **.scf** file:

- `define_clock`
- `define_input_delay`
- `define_output_delay`
- `define_multicycle_path`
- `define_false_path`

All Synplify constraints described above are mapped to SDC commands for the Timing Analyzer.

For syntax and arguments for these commands, refer to the applicable topic in this manual or refer to Synplify Help. For a list of corresponding commands in the Intel Quartus Prime software, refer to the Intel Quartus Prime Help.

#### Related Links

- [Timing-Driven Synthesis Settings](#) on page 1062
- [Intel Quartus Prime Timing Analyzer Documentation](#)



### 19.7.2.1 Individual Clocks and Frequencies

Specify clock frequencies for individual clocks in the Synplify software with the `define_clock` command. This command is passed to the Intel Quartus Prime software with the `create_clock` command.

### 19.7.2.2 Input and Output Delay

Specify input delay and output delay constraints in the Synplify software with the `define_input_delay` and `define_output_delay` commands, respectively. These commands are passed to the Intel Quartus Prime software with the `set_input_delay` and `set_output_delay` commands.

### 19.7.2.3 Multicycle Path

Specify a multicycle path constraint in the Synplify software with the `define_multicycle_path` command. This command is passed to the Intel Quartus Prime software with the `set_multicycle_path` command.

### 19.7.2.4 False Path

Specify a false path constraint in the Synplify software with the `define_false_path` command. This command is passed to the Intel Quartus Prime software with the `set_false_path` command.

## 19.8 Simulation and Formal Verification

You can perform simulation and formal verification at various stages in the design process. You can perform final timing analysis after placement and routing is complete.

If area and timing requirements are satisfied, use the files generated by the Intel Quartus Prime software to program or configure the Intel device. If your area or timing requirements are not met, you can change the constraints in the Synplify software or the Intel Quartus Prime software and rerun synthesis. Intel recommends that you provide timing constraints in the Synplify software and any placement constraints in the Intel Quartus Prime software. Repeat the process until area and timing requirements are met.

You can also use other options and techniques in the Intel Quartus Prime software to meet area and timing requirements, such as WYSIWYG Primitive Resynthesis, which can perform optimizations on your **.vqm** netlist within the Intel Quartus Prime software.

*Note:* In some cases, you might be required to modify the source code if the area and timing requirements cannot be met using options in the Synplify and Intel Quartus Prime software.

## 19.9 Synplify Optimization Strategies

Combining Synplify software constraints with VHDL and Verilog HDL coding techniques and Intel Quartus Prime software options can help you obtain the results that you require.



For more information about applying attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

#### Related Links

- [Design Constraints Support](#) on page 1059
- [Recommended Design Practices Documentation](#) on page 152
- [Timing Closure and Optimization Documentation](#)

### 19.9.1 Using Synplify Premier to Optimize Your Design

Compared to other Synplify products, the Synplify Premier software offers additional physical synthesis optimizations. After typical logic synthesis, the Synplify Premier software places and routes the design and attempts to restructure the netlist based on the physical location of the logic in the Intel device. The Synplify Premier software forward-annotates the design netlist to the Intel Quartus Prime software to perform the final placement and routing. In the default flow, the Synplify Premier software also forward-annotates placement information for the critical path(s) in the design, which can improve the compilation time in the Intel Quartus Prime software.

The physical location annotation file is called `<design name>_plc.tcl`. If you open the Intel Quartus Prime software from the Synplify Premier software user interface, the Intel Quartus Prime software automatically uses this file for the placement information.

The Physical Analyst allows you to examine the placed netlist from the Synplify Premier software, which is similar to the HDL Analyst for a logical netlist. You can use this display to analyze and diagnose potential problems.

### 19.9.2 Using Implementations in Synplify Pro or Premier

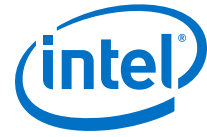
You can create different synthesis results without overwriting the existing results, in the Synplify Pro or Premier software, by creating a new implementation from the Project menu. For each implementation, specify the target device, synthesis options, and constraint files. Each implementation generates its own subdirectory that contains all the resulting files, including `.vqm`, `.scf`, and `.tcl` files, from a compilation of the particular implementation. You can then compare the results of the different implementations to find the optimal set of synthesis options and constraints for a design.

### 19.9.3 Timing-Driven Synthesis Settings

The Synplify software supports timing-driven synthesis with user-assigned timing constraints to optimize the performance of the design.

The Intel Quartus Prime NativeLink feature allows timing constraints that are applied in the Synplify software to be forward-annotated for the Intel Quartus Prime software with an `.scf` file for timing-driven place and route.

The Synplify Synthesis Report File (`.srr`) contains timing reports of estimated place-and-route delays. The Intel Quartus Prime software can perform further optimizations on a post-synthesis netlist from third-party synthesis tools. In addition, designs might contain black boxes or intellectual property (IP) functions that have not been optimized by the third-party synthesis software. Actual timing results are obtained only after the design has been fully placed and routed in the Intel Quartus



Prime software. For these reasons, the Intel Quartus Prime post place-and-route timing reports provide a more accurate representation of the design. Use the statistics in these reports to evaluate design performance.

### Related Links

[Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software on page 1060](#)

## 19.9.3.1 Clock Frequencies

For single-clock designs, you can specify a global frequency when using the push-button flow. While this flow is simple and provides good results, it often does not meet the performance requirements for more advanced designs. You can use timing constraints, compiler directives, and other attributes to help optimize the performance of a design. You can enter these attributes and directives directly in the HDL code. Alternatively, you can enter attributes (not directives) into an **.sdc** file with the SCOPE window in the Synplify software.

Use the SCOPE window to set global frequency requirements for the entire design and individual clock settings. Use the **Clocks** tab in the SCOPE window to specify frequency (or period), rise times, fall times, duty cycle, and other settings. Assigning individual clock settings, rather than over-constraining the global frequency, helps the Intel Quartus Prime software and the Synplify software achieve the fastest clock frequency for the overall design. The `define_clock` attribute assigns clock constraints.

## 19.9.3.2 Multiple Clock Domains

The Synplify software can perform timing analysis on unrelated clock domains. Each clock group is a different clock domain and is treated as unrelated to the clocks in all other clock groups. All clocks in a single clock group are assumed to be related, and the Synplify software automatically calculates the relationship between the clocks. You can assign clocks to a new clock group or put related clocks in the same clock group with the **Clocks** tab in the SCOPE window, or with the `define_clock` attribute.

## 19.9.3.3 Input and Output Delays

Specify the input and output delays for the ports of a design in the **Input/Output** tab of the SCOPE window, or with the `define_input_delay` and `define_output_delay` attributes. The Synplify software does not allow you to assign the  $t_{CO}$  and  $t_{SU}$  values directly to inputs and outputs. However, a  $t_{CO}$  value can be inferred by setting an external output delay; a  $t_{SU}$  value can be inferred by setting an external input delay.

Relationship Between $t_{CO}$ and the Output Delay
$t_{CO} = \text{clock period} - \text{external output delay}$

Relationship Between $t_{SU}$ and the Input Delay
$t_{SU} = \text{clock period} - \text{external input delay}$

When the `syn_forward_io_constraints` attribute is set to 1, the Synplify software passes the external input and output delays to the Intel Quartus Prime software using NativeLink integration. The Intel Quartus Prime software then uses the external delays to calculate the maximum system frequency.

### 19.9.3.4 Multicycle Paths

A multicycle path is a path that requires more than one clock cycle to propagate. Specify any multicycle paths in the design in the **Multi-Cycle Paths** tab of the SCOPE window, or with the `define_multicycle_path` attribute. You should specify which paths are multicycle to prevent the Intel Quartus Prime and the Synplify compilers from working excessively on a non-critical path. Not specifying these paths can also result in an inaccurate critical path reported during timing analysis.

### 19.9.3.5 False Paths

False paths are paths that should be ignored during timing analysis, or should be assigned low (or no) priority during optimization. Some examples of false paths include slow asynchronous resets, and test logic that has been added to the design. Set these paths in the **False Paths** tab of the SCOPE window, or use the `define_false_path` attribute.

## 19.9.4 FSM Compiler

If the FSM Compiler is turned on, the compiler automatically detects state machines in a design, which are then extracted and optimized. The FSM Compiler analyzes state machines and implements sequential, gray, or one-hot encoding, based on the number of states. The compiler also performs unused-state analysis, optimization of unreachable states, and minimization of transition logic. Implementation is based on the number of states, regardless of the coding style in the HDL code.

If the FSM Compiler is turned off, the compiler does not optimize logic as state machines. The state machines are implemented as HDL code. Thus, if the coding style for a state machine is sequential, the implementation is also sequential.

Use the `syn_state_machine` compiler directive to specify or prevent a state machine from being extracted and optimized. To override the default encoding of the FSM Compiler, use the `syn_encoding` directive.

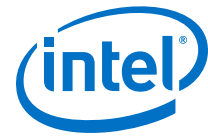
**Table 282.** `syn_encoding` Directive Values

Value	Description
Sequential	Generates state machines with the fewest possible flipflops. Sequential, also called binary, state machines are useful for area-critical designs when timing is not the primary concern.
Gray	Generates state machines where only one flipflop changes during each transition. Gray-encoded state machines tend to be glitches.
One-hot	Generates state machines containing one flipflop for each state. One-hot state machines typically provide the best performance and shortest clock-to-output delays. However, one-hot implementations are usually larger than sequential implementations.
Safe	Generates extra control logic to force the state machine to the reset state if an invalid state is reached. You can use the safe value in conjunction with any of the other three values, which results in the state machine being implemented with the requested encoding scheme and the generation of the reset logic.

### Example 120. Sample VHDL Code for Applying `syn_encoding` Directive

```
SIGNAL current_state : STD_LOGIC_VECTOR (7 DOWNTO 0);
ATTRIBUTE syn_encoding : STRING;
ATTRIBUTE syn_encoding OF current_state : SIGNAL IS "sequential";
```





By default, the state machine logic is optimized for speed and area, which may be potentially undesirable for critical systems. The safe value generates extra control logic to force the state machine to the reset state if an invalid state is reached.

#### 19.9.4.1 FSM Explorer in Synplify Pro and Premier

The Synplify Pro and Premier software use the FSM Explorer to explore different encoding styles for a state machine automatically, and then implement the best encoding based on the overall design constraints. The FSM Explorer uses the FSM Compiler to identify and extract state machines from a design. However, unlike the FSM Compiler, which chooses the encoding style based on the number of states, the FSM Explorer attempts several different encoding styles before choosing a specific one. The trade-off is that the compilation requires more time to analyze the state machine, but finds an optimal encoding scheme for the state machine.

### 19.9.5 Optimization Attributes and Options

#### 19.9.5.1 Retiming in Synplify Pro and Premier

The Synplify Pro and Premier software can retime a design, which can improve the timing performance of sequential circuits by moving registers (register balancing) across combinational elements. Be aware that retimed registers incur name changes. You can retime your design from **Implementation Options** or you can use the `syn_allow_retiming` attribute.

#### 19.9.5.2 Maximum Fan-Out

When your design has critical path nets with high fan-out, use the `syn_maxfan` attribute to control the fan-out of the net. Setting this attribute for a specific net results in the replication of the driver of the net to reduce overall fan-out. The `syn_maxfan` attribute takes an integer value and applies it to inputs or registers. The `syn_maxfan` attribute cannot be used to duplicate control signals. The minimum allowed value of the attribute is 4. Using this attribute might result in increased logic resource utilization, thus straining routing resources, which can lead to long compilation times and difficult fitting.

If you must duplicate an output register or an output enable register, you can create a register for each output pin by using the `syn_useioff` attribute.

#### 19.9.5.3 Preserving Nets

During synthesis, the compiler maintains ports, registers, and instantiated components. However, some nets cannot be maintained to create an optimized circuit. Applying the `syn_keep` directive overrides the optimization of the compiler and preserves the net during synthesis. The `syn_keep` directive is a Boolean data type value and can be applied to wires (Verilog HDL) and signals (VHDL). Setting the value to **true** preserves the net through synthesis.

#### 19.9.5.4 Register Packing

Intel devices allow register packing into I/O cells. Intel recommends allowing the Intel Quartus Prime software to make the I/O register assignments. However, you can control register packing with the `syn_useioff` attribute. The `syn_useioff` attribute is a Boolean data type value that can be applied to ports or entire modules. Setting



the value to **1** instructs the compiler to pack the register into an I/O cell. Setting the value to **0** prevents register packing in both the Synplify and Intel Quartus Prime software.

### 19.9.5.5 Resource Sharing

The Synplify software uses resource sharing techniques during synthesis, by default, to reduce area. Turning off the **Resource Sharing** option on the **Options** tab of the **Implementation Options** dialog box improves performance results for some designs. You can also turn off the option for a specific module with the `syn_sharing` attribute. If you turn off this option, be sure to check the results to verify improvement in timing performance. If there is no improvement, turn on **Resource Sharing**.

### 19.9.5.6 Preserving Hierarchy

The Synplify software performs cross-boundary optimization by default, which causes the design to flatten to allow optimization. You can use the `syn_hier` attribute to override the default compiler settings. The `syn_hier` attribute applies a string value to modules, architectures, or both. Setting the value to **hard** maintains the boundaries of a module, architecture, or both, but allows constant propagation. Setting the value to **locked** prevents all cross-boundary optimizations. Use the **locked** setting with the partition setting to create separate design blocks and multiple output netlists.

By default, the Synplify software generates a hierarchical **.vqm** file. To flatten the file, set the `syn_netlist_hierarchy` attribute to **0**.

### 19.9.5.7 Register Input and Output Delays

Two advanced options, `define_reg_input_delay` and `define_reg_output_delay`, can speed up paths feeding a register, or coming from a register, by a specific number of nanoseconds. The Synplify software attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with the `define_clock` attribute). You can use these attributes to add a delay to paths feeding into or out of registers to further constrain critical paths. You can slow down a path that is too highly optimized by setting this attributes to a negative number.

The `define_reg_input_delay` and `define_reg_output_delay` options are useful to close timing if your design does not meet timing goals, because the routing delay after placement and routing exceeds the delay predicted by the Synplify software. Rerun synthesis using these options, specifying the actual routing delay (from place-and-route results) so that the tool can meet the required clock frequency. Synopsys recommends that for best results, do not make these assignments too aggressively. For example, you can increase the routing delay value, but do not also use the full routing delay from the last compilation.



In the SCOPE constraint window, the registers panel contains the following options:

- **Register**—Specifies the name of the register. If you have initialized a compiled design, select the name from the list.
- **Type**—Specifies whether the delay is an input or output delay.
- **Route**—Shrinks the effective period for the constrained registers by the specified value without affecting the clock period that is forward-annotated to the Intel Quartus Prime software.

Use the following Tcl command syntax to specify an input or output register delay in nanoseconds.

#### Example 121. Input and Output Register Delay

```
define_reg_input_delay {<register>} -route <delay in ns>  
define_reg_output_delay {<register>} -route <delay in ns>
```

#### 19.9.5.8 syn\_direct\_enable

This attribute controls the assignment of a clock-enable net to the dedicated enable pin of a register. With this attribute, you can direct the Synplify mapper to use a particular net as the only clock enable when the design has multiple clock enable candidates.

To use this attribute as a compiler directive to infer registers with clock enables, enter the `syn_direct_enable` directive in your source code, instead of the SCOPE spreadsheet.

The `syn_direct_enable` data type is Boolean. A value of **1** or **true** enables net assignment to the clock-enable pin. The following is the syntax for Verilog HDL:

```
object /* synthesis syn_direct_enable = 1 */ ;
```

#### 19.9.5.9 I/O Standard

For certain Intel devices, specify the I/O standard type for an I/O pad in the design with the **I/O Standard** panel in the Synplify SCOPE window.

The Synplify SDC syntax for the `define_io_standard` constraint, in which the `delay_type` must be either `input_delay` or `output_delay`.

#### Example 122. define\_io\_standard Constraint

```
define_io_standard [-disable|-enable] {<objectName>} -delay_type \  
[input_delay|output_delay] <columnTclName>{<value>} [<columnTclName>{<value>}...]
```

For details about supported I/O standards, refer to the *Synopsys FPGA Synthesis Reference Manual*.

## 19.9.6 Intel-Specific Attributes

You can use the `altera_chip_pin_lc`, `altera_io_powerup`, and `altera_io_opendrain` attributes with specific Intel device features, which are forward-annotated to the Intel Quartus Prime project, and are used during place-and-route.

### 19.9.6.1 altera\_chip\_pin\_lc

Use the `altera_chip_pin_lc` attribute to make pin assignments. This attribute applies a string value to inputs and outputs. Use the attribute only on the ports of the top-level entity in the design. Do not use this attribute to assign pin locations from entities at lower levels of the design hierarchy.

*Note:* The `altera_chip_pin_lc` attribute is not supported for any MAX series device.

In the SCOPE window, set the value of the `altera_chip_pin_lc` attribute to a pin number or a list of pin numbers.

You can use VHDL code for making location assignments for supported Intel devices. Pin location assignments for these devices are written to the output `.tcl` file.

*Note:* The `data_out` signal is a 4-bit signal; `data_out[3]` is assigned to pin 14 and `data_out[0]` is assigned to pin 15.

#### Example 123. Making Location Assignments in VHDL

```
ENTITY sample (data_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
               data_out: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
  ATTRIBUTE altera_chip_pin_lc : STRING;
  ATTRIBUTE altera_chip_pin_lc OF data_out : SIGNAL IS "14, 5, 16, 15";
```

### 19.9.6.2 altera\_io\_powerup

Use the `altera_io_powerup` attribute to define the power-up value of an I/O register that has no set or reset. This attribute applies a string value (**high|low**) to ports with I/O registers. By default, the power-up value of the I/O register is set to **low**.

### 19.9.6.3 altera\_io\_opendrain

Use the `altera_io_opendrain` attribute to specify open-drain mode I/O ports. This attribute applies a boolean data type value to outputs or bidirectional ports for devices that support open-drain mode.

## 19.10 Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including LPMs, device-specific Intel FPGA IP cores, and IP available through the Intel FPGA IP Partners Program (AMPP<sup>SM</sup>). You can use IP cores by instantiating them in your HDL code, or by inferring certain IP cores from generic HDL code.



You can instantiate an IP core in your HDL code with the IP Catalog and configure the IP core with the Parameter Editor, or instantiate the IP core using the port and parameter definition. The IP Catalog and Parameter Editor provide a graphical interface within the Intel Quartus Prime software to customize any available Intel FPGA IP core for the design.

The Synplify software also automatically recognizes certain types of HDL code, and infers the appropriate Intel FPGA IP core when an IP core provides optimal results. The Synplify software provides options to control inference of certain types of IP cores.

#### Related Links

- [Hardware Description Language Support](#) on page 1058
- [Recommended HDL Coding Styles Documentation](#) on page 100
- [About the IP Catalog Online Help](#)

### 19.10.1 Instantiating Intel FPGA IP Cores with the IP Catalog

When you use the IP Catalog and Parameter Editor to set up and configure an IP core, the IP Catalog creates a VHDL or Verilog HDL wrapper file `<output file>.v|vhd` that instantiates the IP core.

The Synplify software uses the Intel Quartus Prime timing and resource estimation netlist feature to report more accurate resource utilization and timing performance estimates, and uses timing-driven optimization, instead of treating the IP core as a “black box.” Including the generated IP core variation wrapper file in your Synplify project, gives the Synplify software complete information about the IP core.

*Note:* There is an option in the Parameter Editor to generate a netlist for resource and timing estimation. This option is not recommended for the Synplify software because the software automatically generates this information in the background without a separate netlist. If you do create a separate netlist `<output file>_syn.v` and use that file in your synthesis project, you must also include the `<output file>.v|vhd` file in your Intel Quartus Prime project.

Verify that the correct Intel Quartus Prime version is specified in the Synplify software before compiling the generated file to ensure that the software uses the correct library definitions for the IP core. The **Quartus Version** setting must match the version of the Intel Quartus Prime software used to generate the customized IP core.

In addition, ensure that the `QUARTUS_ROOTDIR` environment variable specifies the installation directory location of the correct Intel Quartus Prime version. The Synplify software uses this information to launch the Intel Quartus Prime software in the background. The environment variable setting must match the version of the Intel Quartus Prime software used to generate the customized IP core.

#### Related Links

[Specifying the Intel Quartus Prime Software Version](#) on page 1058

#### 19.10.1.1 Instantiating Intel FPGA IP Cores with IP Catalog Generated Verilog HDL Files

If you turn on the `<output file>_inst.v` option on the Parameter Editor, the IP Catalog generates a Verilog HDL instantiation template file for use in your Synplify design. The instantiation template file, `<output file>_inst.v`, helps to instantiate the IP core



variation wrapper file, `<output file>.v`, in your top-level design. Include the IP core variation wrapper file `<output file>.v` in your Synplify project. The Synplify software includes the IP core information in the output `.vqm` netlist file. You do not need to include the generated IP core variation wrapper file in your Intel Quartus Prime project.

### 19.10.1.2 Instantiating Intel FPGA IP Cores with IP Catalog Generated VHDL Files

If you turn on the `<output file>.cmp` and `<output file>_inst.vhd` options on the Parameter Editor, the IP catalog generates a VHDL component declaration file and a VHDL instantiation template file for use in your Synplify design. These files can help you instantiate the IP core variation wrapper file, `<output file>.vhd`, in your top-level design. Include the `<output file>.vhd` in your Synplify project. The Synplify software includes the IP core information in the output `.vqm` netlist file. You do not need to include the generated IP core variation wrapper file in your Intel Quartus Prime project.

### 19.10.1.3 Changing Synplify's Default Behavior for Instantiated Intel FPGA IP Cores

By default, the Synplify software automatically opens the Intel Quartus Prime software in the background to generate a resource and timing estimation netlist for IP cores.

You might want to change this behavior to reduce run times in the Synplify software, because generating the netlist files can take several minutes for large designs, or if the Synplify software cannot access your Intel Quartus Prime software installation to generate the files. Changing this behavior might speed up the compilation time in the Synplify software, but the Quality of Results (QoR) might be reduced.

The Synplify software directs the Intel Quartus Prime software to generate information in two ways:

- Some IP cores provide a "clear box" model—the Synplify software fully synthesizes this model and includes the device architecture-specific primitives in the output `.vqm` netlist file.
- Other IP cores provide a "grey box" model—the Synplify software reads the resource information, but the netlist does not contain all the logic functionality.

*Note:* You need to turn on **Generate netlist** when using the grey box model. For more information, see the Intel Quartus Prime online help.

For these IP cores, the Synplify software uses the logic information for resource and timing estimation and optimization, and then instantiates the IP core in the output `.vqm` netlist file so the Intel Quartus Prime software can implement the appropriate device primitives. By default, the Synplify software uses the clear box model when available, and otherwise uses the grey box model.

#### Related Links

- [Including Files for Intel Quartus Prime Placement and Routing Only](#) on page 1073
- [Synplify Synthesis Techniques with the Intel Quartus Prime Software online training](#)  
Includes more information about design flows using clear box model and grey box model.
- [Generating a Netlist for 3rd Party Synthesis Tools online help](#)



### 19.10.1.4 Instantiating Intellectual Property with the IP Catalog and Parameter Editor

Many Intel FPGA IP cores include a resource and timing estimation netlist that the Synplify software uses to report more accurate resource utilization and timing performance estimates, and uses timing-driven optimization rather than a black box function.

To create this netlist file, perform the following steps:

1. Select the IP core in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Intel Quartus Prime software generates a file `<output file>_syn.v`. This netlist contains the grey box information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file in your Synplify project. Next, include the IP core variation wrapper file `<output file>.v|vhd` in the Intel Quartus Prime project along with your Synplify `.vqm` output netlist.

If your IP core does not include a resource and timing estimation netlist, the Synplify software must treat the IP core as a black box.

#### Related Links

[Including Files for Intel Quartus Prime Placement and Routing Only](#) on page 1073

### 19.10.1.5 Instantiating Black Box IP Cores with Generated Verilog HDL Files

Use the `syn_black_box` compiler directive to declare a module as a black box. The top-level design files must contain the IP port-mapping and a hollow-body module declaration. Apply the `syn_black_box` directive to the module declaration in the top-level file or a separate file included in the project so that the Synplify software recognizes the module is a black box. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates `my_verilogIP.v`, which is a simple customized variation generated by the IP Catalog.

#### Example 124. Sample Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
    input clk;
    output [7:0] count;
    my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule
// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
    input clock;
    output [7:0] q;
endmodule
```

### 19.10.1.6 Instantiating Black Box IP Cores with Generated VHDL Files

Use the `syn_black_box` compiler directive to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port-mapping. Apply the `syn_black_box` directive to the component declaration in the top-level file. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates **my\_vhdlIP.vhd**, which is a simplified customized variation generated by the IP Catalog.

#### Example 125. Sample Top-Level VHDL Code with Black Box Instantiation of IP

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY top IS
  PORT (
    clk: IN STD_LOGIC ;
    count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END top;

ARCHITECTURE rtl OF top IS
  COMPONENT my_vhdlIP
  PORT (
    clock: IN STD_LOGIC ;
    q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
  end COMPONENT;
  attribute syn_black_box : boolean;
  attribute syn_black_box of my_vhdlIP: component is true;
  BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
      clock => clk,
      q => count
    );
  END rtl;
```

### 19.10.1.7 Other Synplify Software Attributes for Creating Black Boxes

Instantiating IP as a black box does not provide visibility into the IP for the synthesis tool. Thus, it does not take full advantage of the synthesis tool's timing-driven optimization. For better timing optimization, especially if the black box does not have registered inputs and outputs, add timing models to black boxes by adding the `syn_tpd`, `syn_tsu`, and `syn_tco` attributes.

#### Example 126. Adding Timing Models to Black Boxes in Verilog HDL

```
module ram32x4(z,d,addr,we,clk);
  /* synthesis syn_black_box syn_tco1="clk->z[3:0]=4.0"
    syn_tpd1="addr[3:0]->[3:0]=8.0"
    syn_tsu1="addr[3:0]->clk=2.0"
    syn_tsu2="we->clk=3.0" */
  output [3:0]z;
  input [3:0]d;
  input [3:0]addr;
  input we;
  input clk;
endmodule
```





The following additional attributes are supported by the Synplify software to communicate details about the characteristics of the black box module within the HDL code:

- `syn_resources`—Specifies the resources used in a particular black box.
- `black_box_pad_pin`—Prevents mapping to I/O cells.
- `black_box_tri_pin`—Indicates a tri-stated signal.

For more information about applying these attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

## 19.10.2 Including Files for Intel Quartus Prime Placement and Routing Only

In the Synplify software, you can add files to your project that are used only during placement and routing in the Intel Quartus Prime software. This can be useful if you have grey or black boxes for Synplify synthesis that require the full design files to be compiled in the Intel Quartus Prime software.

You can also set the option in a script using the `-job_owner par` option.

The example shows how to define files for a Synplify project that includes a top-level design file, a grey box netlist file, an IP wrapper file, and an encrypted IP file. With these files, the Synplify software writes an empty instantiation of “core” in the `.vqm` file and uses the grey box netlist for resource and timing estimation. The files `core.v` and `core_enc8b10b.v` are not compiled by the Synplify software, but are copied into the place-and-route directory. The Intel Quartus Prime software compiles these files to implement the “core” IP block.

### Example 127.

#### Commands to Define Files for a Synplify Project

```
add_file -verilog -job_owner par "core_enc8b10b.v"  
add_file -verilog -job_owner par "core.v"  
add_file -verilog "core_gb.v"  
add_file -verilog "top.v"
```

## 19.10.3 Inferring Intel FPGA IP Cores from HDL Code

The Synplify software uses Behavior Extraction Synthesis Technology (BEST) algorithms to infer high-level structures such as RAMs, ROMs, operators, FSMs, and DSP multiplication operations. Then, the Synplify software keeps the structures abstract for as long as possible in the synthesis process. This allows the use of technology-specific resources to implement these structures by inferring the appropriate Intel FPGA IP core when an IP core provides optimal results.

### Related Links

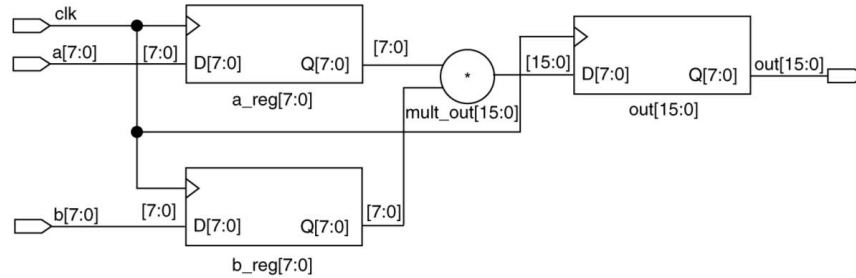
[Recommended HDL Coding Styles Documentation](#) on page 100

### 19.10.3.1 Inferring Multipliers

The figure shows the HDL Analyst view of an unsigned  $8 \times 8$  multiplier with two pipeline stages after synthesis in the Synplify software. This multiplier is converted into an `ALTMULT_ADD` or `ALTMULT_ACCUM` IP core. For devices with DSP blocks, the

software might implement the function in a DSP block instead of regular logic, depending on device utilization. For some devices, the software maps directly to DSP block device primitives instead of instantiating an IP core in the **.vqm** file.

**Figure 340. HDL Analyst View of LPM\_MULT IP Core (Unsigned 8x8 Multiplier with Pipeline=2)**



### 19.10.3.1.1 Resource Balancing

While mapping multipliers to DSP blocks, the Synplify software performs resource balancing for optimum performance.

Intel devices have a fixed number of DSP blocks, which includes a fixed number of embedded multipliers. If the design uses more multipliers than are available, the Synplify software automatically maps the extra multipliers to logic elements (LEs), or adaptive logic modules (ALMs).

If a design uses more multipliers than are available in the DSP blocks, the Synplify software maps the multipliers in the critical paths to DSP blocks. Next, any wide multipliers, which might or might not be in the critical paths, are mapped to DSP blocks. Smaller multipliers and multipliers that are not in the critical paths might then be implemented in the logic (LEs or ALMs). This ensures that the design fits successfully in the device.

### 19.10.3.1.2 Controlling the DSP Block Inference

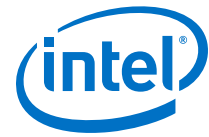
You can implement multipliers in DSP blocks or in logic in Intel devices that contain DSP blocks. You can control this implementation through attribute settings in the Synplify software.

### 19.10.3.1.3 Signal Level Attribute

You can control the implementation of individual multipliers by using the `syn_multstyle` attribute as shown in the following Verilog HDL code (where `<signal_name>` is the name of the signal ):

```
<signal_name> /* synthesis syn_multstyle = "logic" */;
```

The `syn_multstyle` attribute applies to wires only; it cannot be applied to registers.



**Table 283. DSP Block Attribute Setting in the Synplify Software**

Attribute Name	Value	Description
syn_multstyle	lpm_mult	LPM function inferred and multipliers implemented in DSP blocks.
	logic	LPM function not inferred and multipliers implemented as LEs by the Synplify software.
	block_mult	DSP IP core is inferred and multipliers are mapped directly to DSP block device primitives (for supported devices).

**Example 128. Signal Attributes for Controlling DSP Block Inference in Verilog HDL Code**

```

module mult(a,b,c,r,en);
  input [7:0] a,b;
  output [15:0] r;
  input [15:0] c;
  input en;
  wire [15:0] temp /* synthesis syn_multstyle="logic" */;

  assign temp = a*b;
  assign r = en ? temp : c;
endmodule

```

**Example 129. Signal Attributes for Controlling DSP Block Inference in VHDL Code**

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity onereg is port (
  r : out std_logic_vector (15 downto 0);
  en : in std_logic;
  a : in std_logic_vector (7 downto 0);
  b : in std_logic_vector (7 downto 0);
  c : in std_logic_vector (15 downto 0);
);
end onereg;

architecture beh of onereg is
  signal temp : std_logic_vector (15 downto 0);
  attribute syn_multstyle : string;
  attribute syn_multstyle of temp : signal is "logic";

begin
  temp <= a * b;
  r <= temp when en='1' else c;
end beh;

```

**19.10.3.2 Inferring RAM**

When a RAM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device primitives instead of instantiating an IP core in the .vqm file.

Follow these guidelines for the Synplify software to successfully infer RAM in a design:

- The address line must be at least two bits wide.
- Resets on the memory are not supported. Refer to the device family documentation for information about whether read and write ports must be synchronous.
- Some Verilog HDL statements with blocking assignments might not be mapped to RAM blocks, so avoid blocking statements when modeling RAMs in Verilog HDL.

For some device families, the `syn_ramstyle` attribute specifies the implementation to use for an inferred RAM. You can apply the `syn_ramstyle` attribute globally to a module or a RAM instance, to specify `registers` or `block_ram` values. To turn off RAM inference, set the attribute value to `registers`.

When inferring RAM for some Intel device families, the Synplify software generates additional bypass logic. This logic is generated to resolve a half-cycle read/write behavior difference between the RTL and post-synthesis simulations. The RTL simulation shows the memory being updated on the positive edge of the clock; the post-synthesis simulation shows the memory being updated on the negative edge of the clock. To eliminate bypass logic, the output of the RAM must be registered. By adding this register, the output of the RAM is seen after a full clock cycle, by which time the update has occurred, thus eliminating the need for bypass logic.

For devices with TriMatrix memory blocks, disable the creation of glue logic by setting the `syn_ramstyle` value to `no_rw_check`. Set `syn_ramstyle` to `no_rw_check` to disable the creation of glue logic in dual-port mode.

### Example 130.

#### VHDL Code for Inferred Dual-Port RAM

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0)
      wr_addr, rd_addr: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
      we: IN STD_LOGIC;
      clk: IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL mem: Mem_Type;
SIGNAL addr_reg: STD_LOGIC_VECTOR (6 DOWNTO 0);
BEGIN
  data_out <= mem (CONV_INTEGER(rd_addr));
  PROCESS (clk, we, data_in) BEGIN
    IF (clk='1' AND clk'EVENT) THEN
      IF (we='1') THEN
        mem(CONV_INTEGER(wr_addr)) <= data_in;
      END IF;
    END IF;
  END PROCESS;
END ram_infer;
```

### Example 131.

#### VHDL Code for Inferred Dual-Port RAM Preventing Bypass Logic

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
```



```
ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      data_in : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      wr_addr, rd_addr : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
      we : IN STD_LOGIC;
      clk : IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL mem : Mem_Type;
SIGNAL addr_reg : STD_LOGIC_VECTOR (6 DOWNTO 0);
SIGNAL tmp_out : STD_LOGIC_VECTOR (7 DOWNTO 0); --output register

BEGIN
    tmp_out <= mem (CONV_INTEGER (rd_addr));
    PROCESS (clk, we, data_in) BEGIN
        IF (clk='1' AND clk'EVENT) THEN
            IF (we='1') THEN
                mem(CONV_INTEGER(wr_addr)) <= data_in;
            END IF;
            data_out <= tmp_out; --registers output preventing
                                -- bypass logic generation
        END IF;
    END PROCESS;
END ram_infer;
```

### 19.10.3.3 RAM Initialization

Use the Verilog HDL `$readmemb` or `$readmemh` system tasks in your HDL code to initialize RAM memories. The Synplify compiler forward-annotates the initialization values in the **.srs** (technology-independent RTL netlist) file and the mapper generates the corresponding hexadecimal memory initialization (**.hex**) file. One **.hex** file is created for each of the `altsyncram` IP cores that are inferred in the design. The **.hex** file is associated with the `altsyncram` instance in the **.vqm** file using the `init_file` attribute.

The examples show how RAM can be initialized through HDL code, and how the corresponding **.hex** file is generated using Verilog HDL.

#### Example 132.

#### Using `$readmemb` System Task to Initialize an Inferred RAM in Verilog HDL Code

```
initial
begin
    $readmemb("mem.ini", mem);
end
always @(posedge clk)
begin
    raddr_reg <= raddr;
    if(we)
        mem[waddr] <= data;
end
```

#### Example 133.

#### Sample of **.vqm** Instance Containing Memory Initialization File

```
altsyncram mem_hex( .wren_a(we),.wren_b(GND),...);

defparam mem_hex.lpm_type = "altsyncram";
defparam mem_hex.operation_mode = "Dual_Port";
...
defparam mem_hex.init_file = "mem_hex.hex";
```



### 19.10.3.4 Inferring ROM

When a ROM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device atoms instead of instantiating an IP core in the **.vqm** file.

Follow these guidelines for the Synplify software to successfully infer ROM in a design:

- The address line must be at least two bits wide.
- The ROM must be at least half full.
- A CASE or IF statement must make 16 or more assignments using constant values of the same width.

### 19.10.3.5 Inferring Shift Registers

The Synplify software infers shift registers for sequential shift components so that they can be placed in dedicated memory blocks in supported device architectures using the ALTSHIFT\_TAPS IP core.

If necessary, set the implementation style with the `syn_srlstyle` attribute. If you do not want the components automatically mapped to shift registers, set the value to `registers`. You can set the value globally, or on individual modules or registers.

For some designs, turning off shift register inference improves the design performance.

## 19.11 Document Revision History

Table 284. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	<ul style="list-style-type: none"><li>• Implemented Intel rebranding.</li></ul>
2016.05.03	16.0.0	<ul style="list-style-type: none"><li>• Removed support for NativeLink synthesis in Pro Edition</li></ul>
2015.11.02	15.1.0	<ul style="list-style-type: none"><li>• Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li></ul>
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
November 2013	13.1.0	Dita conversion. Restructured content.
June 2012	12.0.0	Removed survey link.
November 2011	10.1.1	Template update.
December 2010	10.1.0	<ul style="list-style-type: none"><li>• Changed to new document template.</li><li>• Removed Classic Timing Analyzer support.</li><li>• Removed the "altera_implement_in_esb or altera_implement_in_eab" section.</li><li>• Edited the "Creating a Intel Quartus Prime Project for Compile Points and Multiple .vqm Files" on page 14–33 section for changes with the incremental compilation flow.</li><li>• Edited the "Creating a Intel Quartus Prime Project for Multiple .vqm Files" on page 14–39 section for changes with the incremental compilation flow.</li><li>• Editorial changes.</li></ul>
July 2010	10.0.0	<ul style="list-style-type: none"><li>• Minor updates for the Intel Quartus Prime software version 10.0 release.</li></ul>
November 2009	9.1.0	<ul style="list-style-type: none"><li>• Minor updates for the Intel Quartus Prime software version 9.1 release.</li></ul>
<i>continued...</i>		



Date	Version	Changes
March 2009	9.0.0	<ul style="list-style-type: none"> <li>• Added new section “Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration” on page 14–14.</li> <li>• Minor updates for the Intel Quartus Prime software version 9.0 release.</li> <li>• Chapter 10 was previously Chapter 9 in software version 8.1.</li> </ul>
November 2008	8.1.0	<ul style="list-style-type: none"> <li>• Changed to 8-1/2 x 11 page size</li> <li>• Changed the chapter title from “Synplicity Synplify &amp; Synplify Pro Support” to “Synopsys Synplify Support”</li> <li>• Replaced references to Synplicity with references to Synopsys</li> <li>• Added information about Synplify Premier</li> <li>• Updated supported device list</li> <li>• Added SystemVerilog information to Figure 14–1</li> </ul>
May 2008	8.0.0	<ul style="list-style-type: none"> <li>• Updated supported device list</li> <li>• Updated constraint annotation information for the Timing Analyzer</li> <li>• Updated RAM and MAC constraint limitations</li> <li>• Revised Table 9–1</li> <li>• Added new section “Changing Synplify’s Default Behavior for Instantiated Altera Megafunctions”</li> <li>• Added new section “Instantiating Intellectual Property Using the MegaWizard Plug-In Manager and IP Toolbench”</li> <li>• Added new section “Including Files for Intel Quartus Prime Placement and Routing Only”</li> <li>• Added new section “Additional Considerations for Compile Points”</li> <li>• Removed section “Apply the LogicLock Attributes”</li> <li>• Modified Figure 9–4, 9–43, 9–47. and 9–48</li> <li>• Added new section “Performing Incremental Compilation in the Intel Quartus Prime Software”</li> <li>• Numerous text changes and additions throughout the chapter</li> <li>• Renamed several sections</li> <li>• Updated “Referenced Documents” section</li> </ul>

### Related Links

#### Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.