GenSys®
User Guide

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

©2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

About This User Guide xxiv

Customer Support. xxvi

1. Using the GenSys Tcl Commands

GenSys Concepts. 1-2
- GenSys Objects. 1-2
  - Designs. 1-2
  - Components 1-2
  - Interfaces 1-2
  - Design Configurations 1-3
- Formats for Saving GenSys Objects 1-4
- Supported Formats During Object Import and Export 1-4
- Using VLNV to Identify an Object 1-4
- Stop Modules. 1-5
- GenSys Files 1-5
  - Configuration File 1-5
  - The gensysrc File 1-6

Getting Started with GenSys 1-7
- Setting the GenSys Path 1-7
  - csh 1-7
  - sh, bash 1-8
- Setting the GenSys License 1-8
  - csh 1-8
  - sh, bash 1-8
- Setting the Configuration File 1-8

Setting the Directory Structure 1-8
Setting the Library Path 1-9
Setting the Output Directory 1-9
Setting the Verilog and VHDL Directories 1-10
Setting the Report Directory 1-10
Setting the Log Directory 1-10
Directory to Store Internally-Generated Files 1-11

Invoking GenSys 1-11
Invoking GenSys in the GUI Mode 1-11
Invoking GenSys in the Batch Mode 1-11
Displaying Debug Messages 1-12

Introducing GenSys GUI 1-12
The Canvas View 1-12
The Tabular View 1-13
Work Area in the GUI Mode 1-14

Closing GenSys 1-14

GenSys Exit Status 1-14

2. Introducing GenSys Flows

The IP Packaging Flow 2-2
Using IP-XACT in GenSys 2-2
Loading the IP-XACT Data 2-2
Translation of IP-XACT Files 2-3
Specifying the Default Version of IP-XACT Files 2-4
Saving the IP-XACT Data 2-4
Validating the IP-XACT Files 2-4
Using Direction and Bit-width Constraints 2-4
Mapping Single Logical Port to Multiple Actual Ports 2-6
Using Phantom Ports 2-6
Using Monitor Interfaces 2-7
Using ALDA Ports 2-7
Validating the IP-XACT Data by Running the SCR Checks 2-8
Defining the Scope of SCR Checks 2-22
Viewing the IP-XACT Data in the IP-XACT Viewer Pane 2-22
Additional Information on the IP-XACT Support 2-23
Tasks in the IP Packaging Flow 2-26
Setting the Library Path 2-27
Loading the RTL of an IP 2-27

Updating Interface Definitions. 2-27
Specifying Packaging Options 2-27
Generating the Packaged View 2-28
Checking the RTL Against the Packaged View. 2-28
Performing Incremental Packaging. 2-28
Performing Hierarchical Packaging 2-29
Viewing Reports 2-29

The RTL Restructuring Flow 2-29
The Need to Modify the Design Hierarchy. 2-29
Tasks in the RTL Restructuring Flow 2-30
Loading the RTL of an IP 2-30
Moving Instances 2-31
Adding a Hierarchy 2-32
Dissolving a Hierarchy 2-35
Grouping 2-36
Removing Floating Ports After Hierarchy Manipulation 2-37
Reverting Hierarchy Manipulation Operations 2-38
Optimizing Ports Generated on Subsystem Boundaries 2-38
Viewing Library Cells in a Hierarchy 2-38
Comparing Original and New Hierarchies 2-39
Cloning Instances 2-40
Adding Clones in the Hierarchy of Instances of the Specified Design Masters 2-43
Adding Planes. 2-45
Performing Equivalence Checking 2-45
Checking the Design Health 2-45
Debugging and Analyzing the Design. 2-49
Generating RTL 2-50
Viewing Reports 2-50

The Structured Assembly Flow 2-50
Tasks in the Structured Assembly Flow 2-50

3. Creating Libraries

Prerequisites for Library Creation 3-2
Setting VLNV 3-2
Setting a Library Path 3-3
Specifying a Default Save Format 3-3

Building Interfaces and Interface Connections 3-3
Creating Interfaces 3-4

Using the GUI Menu to Create Interfaces . . . 3-4
Using the Canvas View to Create Interfaces . . . 3-4
Using the Canvas View to Create Interfaces On-the-Fly . . . 3-4
Using the Tabular View to Create Interfaces . . . 3-6
Using Tcl to Create Interfaces . . . 3-7
Using MS Excel to Create Interfaces . . . 3-8
Using the Interfaces Suggested in GenSys . . . 3-8
Adding Ports On Interfaces . . . 3-8
Using the Canvas View to Add Interface Ports . . . 3-9
Using the Tabular View to Add Interface Ports . . . 3-9
Using Tcl to Add Interface Ports . . . 3-10
Creating Interface Instances . . . 3-10
Mapping Interfaces . . . 3-10
Advantages of Mapping Interfaces . . . 3-11
Using the Interface Matching Dialog to Map Interfaces . . . 3-12
Using Tcl to Map Interfaces . . . 3-17
Using the Tabular View to Map Interfaces . . . 3-18
Using Rule-Based Interface Mapping . . . 3-18
Important Guidelines for Defining Interfaces . . . 3-29
Building Components . . . 3-30
Creating Components . . . 3-30
Using the Canvas View to Create Components . . . 3-30
Using the Tabular View to Create Components . . . 3-39
Using Tcl to Create Components . . . 3-42
Using MS Excel to Create Components . . . 3-43
Importing Components . . . 3-43
Handling VHDL Designs During RTL Import . . . 3-44
Handling SystemVerilog IPs During RTL Import . . . 3-44
Mapping Interfaces on Components . . . 3-44
Building Designs . . . 3-44
Working with Attributes and Parameters . . . 3-44
Working with Attributes . . . 3-44
Assigning Attributes to Objects . . . 3-45
Retrieving Attribute Values . . . 3-47
Working with Parameters . . . 3-47
Assigning Parameters . . . 3-47
Managing Dependant Parameters . . . 3-49
Working with 3-State Values . . . 3-50

4. Performing RTL Import

Importing RTL Through GUI 4-3
Configuring Stop Modules and Macros 4-5
Configuring Sub-Systems as Stop Modules 4-5
Configuring Macros. 4-9
Importing RTL by Using Tcl Command 4-9
Merging Functionalities of Existing and Imported Object 4-9
Performing Force RTL Import 4-10
Inferring Glue Logic During RTL Import 4-11
Importing Glue Logic 4-11
Creating Separate Glue Components 4-11
Editing Glue Logic 4-12
Handling Complex Port Types 4-12
Example of Inferring Glue Logic 4-13
Inferring Interfaces on Imported RTL 4-13
Checks Performed While Inferring Interfaces 4-14
Inferring IP-XACT Interfaces 4-14
Inferring Interface Connections 4-15
Preserving Net Names after Updating the Imported RTL 4-15
Examples 4-15
Example 1. 4-15
Example 2 - Preserving Net Names when Sink Side of a Connection is Partially Connected 4-16
Example 3 - Preserving Net Names in Case of Tieoff Connectivity 4-16
Handling Verilog Pre Processor Directives 4-17
Loading Configurable Objects 4-17
Enabling and Disabling Macros During RTL Import. 4-18
Creating Configurable Objects. 4-20
Reporting Unsupported Constraints in GenSys 4-21
Unsupported Constructs in GenSys 4-21
Unsupported Constructs in the VHDL Flow 4-22
Unsupported Constructs in the Verilog Flow. 4-22
Unsupported Constructs in the SystemVerilog Flow. 4-24
The RTL Profiler Report. 4-24
Generating the RTL Profiler Report 4-24

The RTL constructs unsupported in GenSys 4-25
Impact of GenSys switches on RTL constructs 4-25
Information on various constructs present in the RTL 4-25
Examples of Constructs to Help Better Understand the RTL. 4-26
Example on the Usage of ‘ifdef and ‘define 4-27
Example on Generics and Local/Complex/String Parameters 4-27
Example of Parameterized Multi-Dimensional Ports/Wires 4-27
Example of Complex User-Defined Ports. 4-27
Example on the Presence of Generate Blocks. 4-27
Example on the Presence of Array-of-Instances 4-28
Example of Partial Assignments in Glue/Generate Logic. 4-28
Example Of VHDL Modules of Complex Types 4-28
Example of Local ‘define Constructs 4-28
Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations 4-28
Example of Escaped Identifiers 4-28
Example of Pragmas in the Same Line as RTL Constructs 4-28
Example of defparam/Usage of real Keyword in AMS 4-29
Example of Multiple Definitions of the Same Macro. 4-29
Example of Local `undef Constructs. 4-29
Examples of `ifdef on Partial Expressions/Statements. 4-29
Example of RTL Code for Unsynthesizable Constructs in Same Line as Macros 4-30
Presence of Multiple `ifdef Branches in the Same Line 4-30
Example of `ifdef on Complete Module. 4-30
Example of ifdef on `define, `undef, `timescale 4-30
Example of `ifdef on Assertions, Initial Blocks, and Unsynthesizable Constructs 4-30

5. Managing Connections

Understanding Connection Types in GenSys. 5-2
Adhoc Connections 5-2
Ad-hoc Tieoffs 5-2
Interface Connections 5-3
Unconnected 5-4
Exported Interface 5-4
Default Open 5-4
Default Tieoff 5-5
Understanding Connection Precedence. 5-5

Validating Ad-hoc and Interface Connections 5-6

Creating Connections 5-6
Using the Connection Builder Dialog to Create Connections 5-6
Invoking the Connection Builder Dialog 5-7
Details of the Connection Builder Dialog 5-8
Creating a New Connection 5-9
Modifying an Existing Connection. 5-13
Deleting a Connection 5-14
Using the Autoconnect Feature to Create Connections. 5-15
Generating Automatic Connections by Using Rules 5-15
Generating Automatic Connections by Using Tcl Commands 5-20
Using the Canvas View to Create Connections. 5-26
Using Tcl to Create Connections 5-26

Controlling Net Names while Creating Connections. 5-26

Managing Overridden Connections 5-26
Examples of Overridden Connections 5-26

Generating Tie-off Connections 5-27
Using GUI to Generate Tie-Off Connections 5-28
Using Tcl to Generate Tie-Off Connections 5-28

Generating Splice Connections 5-29

Specifying Vector Port Alignment 5-29

Specifying Bit Endianness 5-29

Debugging Connections in Schematic 5-30

Support for Connectivity Verification. 5-31
Enable the Generation of SGDC Constraints 5-32
Supported Connection Addition and Deletion Commands. 5-32
Type of SGDC Constraints Generated 5-32
Example. 5-34
TCL Script - test.tcl 5-35
Generated SGDC Constraints File - verify_connections.sgdc 5-35
Limitations 5-35

6. Building SoCs

Understanding the Top-Down Approach 6-2

Understanding the Bottom-Up Approach 6-2

Building Designs from Scratch 6-2
Specifying a Default Save Format for the Design 6-3
Creating a New Design 6-3
Using the GUI Menu Option to Create Designs 6-3
Using the Canvas View to Create Designs. 6-4
Using Tcl to Create Designs 6-4
Using MS Excel to Create Designs. 6-4
Adding Ports on the Design 6-5
Using the Canvas View to Add Ports 6-6
Using the Tabular View to Add Ports 6-6
Using Tcl to Add Ports. 6-6
Adding Instances in the Design 6-7
Using the Canvas View to Add Instances 6-7
Using the Tabular View to Add Instances 6-7
Using Tcl to Add Instances 6-8
Modifying Instance Parameters 6-8
Using the Right-Click Option in the Design Browser tab 6-9
Using the Property Editor Pane in the Canvas View 6-9
Using Tcl 6-9
Adding Glue Logic 6-9
Creating Connections 6-9
Controlling Port and Net Names of Connections 6-9
Controlling Net Names 6-9
Controlling Port Names. 6-9
Exporting Interfaces and Ports on the Design 6-10
Using the Canvas View to Export Pins and Interfaces 6-11
Using the Connection Builder Dialog to Export Interfaces and Ports 6-11
Using Tcl to Export Interfaces and Ports. 6-11
Updating Master 6-11
Using GUI to Update to the New Version of the IP 6-11
Using Tcl to Update to the New Version of the IP 6-11
Updating the Design Hierarchy 6-11
Saving the Design 6-12
Viewing Reports. 6-12
Checking Design Health 6-12
Generating the RTL 6-12

Building Derivative Designs 6-12

Loading Existing Designs 6-13
Loading the Existing XML of the Design 6-14
Loading the Design from the Library Browser 6-14
Loading the Design RTL 6-14
Importing MS Excel Containing Design Data 6-14
Abstracting Interface Connections 6-14
Handling Glue Logic During RTL Import 6-14
Solidifying Glue Instances and Generate Blocks 6-15
Solidifying Individual Glue Logic from a Glue Group 6-17
Solidifying a Glue Group 6-20
Files Generated for Solidified Glue Logic 6-22
Updating Instance Details 6-23
Managing Connections 6-23
Customizing Ports and Net Name 6-23
Updating the Master of an IP 6-23
Performing RTL Health Check 6-23
Performing Hierarchy Manipulations 6-23
Comparing Different Versions of an Object 6-24
Introducing the Difference Merging Tool 6-24
Invoking the Difference Merging Tool 6-25
Using the Difference Merging Tool 6-26
Advantages of Difference Merging Tool 6-28
Uniquifying Instances 6-28
Using GUI to Uniquify Instances 6-28
Using Tcl to Uniquify Instances 6-29
Configuring Macros 6-30
Configuring Macros During RTL Import 6-30
Configuring Macros During Design Assembly 6-31
Controlling Visibility 6-32
Working with Generate Blocks 6-36
Swapping Memories 6-37
Example of RTL Containing Generic Memories 6-38
Types of Memory Swapping 6-39
Propagating Memory Pins Across Hierarchies After Memories Swap 6-40
Swapping Memories Inside Generate Blocks 6-40
Example of Memory Swapping 6-41
Editing generate Blocks and Array-Of-Instances 6-45
Prerequisites to Edit Generate Blocks and Array-Of-Instances 6-45
Multi-Instance 6-45

array-of-instances *6-45*
generate-for 6-46
generate if *6-48*
Updating the Master of Instances 6-49
Example of Updating the Master of a generate-for Multi-Instance 6-50
Adding Tieoff Connections to Instances 6-52
Example of Adding a Tieoff Connection from a Multi-Instance 6-53
Deleting Connections from Instances 6-56
Exporting Instance Pins 6-57
Example of Exporting a Multi-Instance Pin 6-57
Exporting Complex Multi-Instance Pins 6-60
Example of Specifying RTL Expression in -loop_expr 6-60

7. Generating RTL

Adding Comments in the RTL 7-3
Adding Parameter Declaration in the RTL 7-3
Adding Connections Description in the RTL 7-3
Adding the Delta Delay Information in the RTL 7-3
Adding Header in the RTL 7-4
The Header Template File 7-4
Adding Verilog Timescale Directives in the RTL 7-5
Adding Component Declaration in the RTL (VHDL) 7-5
Generating Optimized Nets in the RTL 7-5
Handling Holes in the Generated RTL 7-6
Adding VSS and VCC Information in the Generated RTL 7-6
Adding Black Box Information in the Generated RTL 7-7
Preserving MSB and LSB in the Generated RTL 7-7
Specifying the Format of VHDL Architectures in the Generated RTL 7-7
Specifying VHDL Generation Style 7-7
Customizing Net Names in the Generated RTL 7-8
Using the spyglass.f File 7-8
Using the sources.f File 7-9

Selective RTL Generation 7-9
Selective RTL Generation of RTL Modules and Entities 7-9
Modules Eligible for Selective RTL Generation 7-10
Selective RTL Generation Internal to Modules and Entities 7-10
Enabling the Selective RTL Generation Flow 7-10
Example of Selective RTL Generation 7-11
RTL Generated for the Objects Having Same Name 7-13

8. GenSys GUI Modes

Using the Canvas View 8-2
Performing Various Actions in the Canvas View 8-2
Entering into the Edit Mode 8-3
Loading Objects from Files 8-4
Creating New Objects 8-5
Deleting Objects 8-6
Adding Ports 8-6
Exporting Pins an Interfaces to the Design Boundary 8-7
Creating Direct Connections Between Objects 8-10
Creating Connections with Additional Details 8-13
Specifying a Net Name for a Connection 8-17
Deleting Connected Nets 8-18
Adding Glue Logic in a Design 8-18
Inferring Interfaces Based on Suggestions from the Tool 8-22
Displaying the Master of an Instance 8-24
Refreshing the Canvas View 8-24
Undocking the Schematic 8-24
Searching Objects in the Schematic 8-25
Creating New Objects 8-26
Modifying RTL of an Object 8-28
Creating a Net Bundle 8-28
Working with Planes 8-29
Working with Virtual Groups 8-38
Updating Color Settings and Miscellaneous Settings 8-46
Expanding or Collapsing Interface Details 8-50
Changing Magnification in Schematic 8-52
Creating Focus on the Selected Object 8-52
Viewing Only Unconnected Ports and Pins 8-53
Navigating between Unconnected Ports and Pins 8-54
Modifying Configurability 8-55
Viewing Connectivity between Two Instances 8-55

Performing Undo and Redo 8-56
Using the Property Editor Pane 8-56
Modifying Properties of an Object. 8-57
Viewing Property Details of an Object 8-62
Using the Tabular View 8-64
Understanding the Tabular View 8-64
Understanding Tab Coloring 8-65
Understanding Tabs of a Particular Object Type 8-65
Working in the Tabular View 8-66
Inserting Rows 8-66
Moving Through Cells. 8-66
Deleting Rows. 8-67
Copying/Pasting Data 8-67
In Non-editable Drop-down Lists 8-68
Specifying Live Formula using Tcl Expressions 8-68
Specifying Live Formula using Perl Expressions 8-72
Using Tabular View Right-Click Menu Options 8-75
Using Table Column Menu Options 8-81
Using Tab Menu Options 8-84
Using Keyboard Shortcuts 8-85
Working in Different Modes 8-86
Using Procedures while Working with Modes. 8-87

9. Customizing GenSys

Creating Custom Generators 9-2
Creating Custom Reports 9-2
Modifying SCR Checks 9-2
Encrypting Perl and Tcl Files. 9-2
Files Generated After Encryption. 9-3
Using Encrypted Files - Decryption 9-3
Using Encrypted Perl Files 9-3
Using Encrypted Tcl Files 9-4

10. Spreadsheet Methodology in GenSys Assembly

The Template MS Excel File 10-3
Importing Data From MS Excel File in GenSys 10-3

Exporting Data from GenSys in the MS Excel File 10-4

Worksheets In the Template Spreadsheet File 10-4
Specifying Comments in Worksheets 10-4
Specifying Rows to Ignore in a Worksheet 10-4
The README Worksheet 10-5
The TopDesignVLNV Worksheet 10-5
The Instances Worksheet 10-6
Specifying Parameters to be Overridden 10-8
Specifying the Hierarchy to Place an Instance in the Instances Worksheet 10-8
The Connections Worksheet 10-9
The Components Worksheet 10-10
The Interface Worksheet 10-12
The InterfaceMapping Worksheet 10-14

11. GenSys GUI Details

Library Browser Tab 11-4
Viewing Library Cells in the Library Browser 11-5
Filtering Objects in the Library Browser 11-5
Library Browser Tab Menu Options 11-7
Add Lib File 11-7
Add/Remove Files With VLNV 11-7
Remove From Library 11-8
Edit Search Path 11-8
Scan Directory 11-8
Refresh Library 11-9
Replace IP 11-9
View Source File 11-11
Find .. Ctrl+F 11-11

Design Browser Tab 11-11
Library Cells in the Design Browser 11-12
Coloring Convention for Instantiated Interfaces in Design Browser 11-13
Design Browser Tab Menu Options 11-13
Open Master 11-14
View XML 11-14
View RTL Source 11-14
Read Master Component 11-14
Reconfigure Tie off Connections 11-14
Reconfigure Instance 11-14

Update Masters 11-15
Delete 11-16
Load to IS 11-16
Add to IS 11-16
Find 11-16
Make Leaf Component 11-17
Group 11-17
Ungroup 11-24

IP-XACT Viewer Pane 11-26
Showing/Hiding the IP-XACT Viewer Pane 11-27

Command/Log Pane 11-28
Viewing Messages in Different Formats 11-30
Viewing Messages in a Tree View Format 11-30
Viewing Messages in a Flat View Format 11-31
Viewing Messages in Different Modes 11-32
Waiving and Un-waiving Messages 11-32
Waiving Messages 11-32
Un-Waiving Messages 11-33
Storing Waived Messages of the Last Session 11-33
Setting Log Filters 11-33
Specifying Commands 11-33

Modular Schematic Window 11-34
Displaying the Modular Schematic Window 11-34
Navigating in the Schematic Using Click-Drag Combinations 11-35
Using the Modular Schematic Window Menu Bar 11-36
File 11-36
Edit 11-39
Tools 11-39

Incremental Schematic Window 11-41
Adding Objects in Incremental Schematic 11-41
Load to IS 11-41
Add to IS 11-43
Viewing Connections in Incremental Schematic Window 11-44
Navigating through the Incremental Schematic Window 11-45
Specifying Incremental Schematic Options 11-45
Right-Click Menu Options on the Blank Area of Incremental Schematic 11-46
Right-Click Menu Options on an Instance in Incremental Schematic 11-47

Right-Click Menu Options on a Pin in Incremental Schematic 11-48
Finding Objects in Incremental Schematic 11-49
Statistics Window 11-51
Menu Bar 11-53
File Menu 11-53
File > New. 11-53
File > Open 11-54
File > Save 11-54
File > Export 11-54
File> Save All 11-55
File > Export All. 11-56
File > Save Session 11-56
File > Restore Session 11-56
File > Close. 11-57
File > Close All 11-57
File > Print Schematic. 11-57
File > Load RTL 11-57
File > Set Working Directory 11-58
File > Set Output Directory 11-58
File > Set XML Directory. 11-58
File > Set Save Template 11-58
File > Set Reports Directory 11-59
File > Set Netlist Directory 11-59
File > Exit 11-59
Edit Menu. 11-59
Menu Options in the Tabular View 11-60
Menu Options in the Canvas View 11-62
View Menu 11-63
Goto Next Error. 11-63
Goto Previous Error 11-63
Views 11-63
Library Browser Menu 11-64
Add 11-64
Add/Remove Files With VLNV 11-64
Remove From Library 11-66
Edit Search Path. 11-66
Scan Directory 11-66
Generator Menu. 11-66
View Generated Files 11-67
Template 11-67

Generate VHDL 11-67
Generate Verilog 11-67
Tools Menu 11-67
Connection Builder 11-68
Configurability/Visibility 11-68
Elaborate 11-68
Incremental Schematic 11-69
Setup Rules 11-69
Interface Matching Widget 11-70
Tcl Editor 11-70
Hierarchy Manipulation 11-71
Options 11-72
Reports Menu 11-73
Specifying a Directory to Save Reports 11-74
Design Summary 11-74
Interface Summary 11-75
Connection Summary 11-77
Suggested Interfaces 11-81
Diff Report 11-82
Diff Report (Details) 11-86
Packaging Report 11-88
ERC Report 11-88
RTLHC Report 11-89
Check RTL 11-90
Help Menu 11-93
GenSys User Guide 11-94
Tcl Commands Reference 11-94
Generating Output Guide 11-94
Customization Guide 11-94
Schema Guide 11-94
Assembly Methodology Guide 11-94
Registers Methodology Guide 11-94
Generating IO Logic Guide 11-94
Release Notes 11-94
KPNS 11-94
About 11-95
Toolbar 11-95

12. Using GenSys in the Batch Mode

-batch 12-3

-autortl2mrs <files> -dir <destdir> 12-3
-mrs2autortl <files> -dir <destdir> 12-3
-I <dirName> 12-3
-schema <dirName> | <file> 12-3
-includeAutoRTLSchema 12-3
-vendorExtnSchema <vendor-name> 12-3
-useSupplyName 12-4
-batchSch 12-4
-dumptcl <files> -dir <destdir> 12-4
-dumpxml <file> -dir <destdir> 12-4
-dumpspirit <file> -dir <destdir> 12-4
-dumpsingletcl 12-4
-debug 12-5
-log_dir <dir> 12-5
--perflog 12-5
-outdir <dir> 12-5
-XML_DIR <dir> 12-5
-report_dir <dir> 12-5
-tcl <files> 12-6
-help 12-6
-stopTclInterpOnErrors 12-6
-strict 12-6
-trace 12-6
-use_old_elab_flow 12-6
-version 12-7
-64bit 12-7
-syncSpirit 12-7
-dumpsingletcl <files> -dir <destdir> 12-7

-product 12-7
-rcfile <config-file> 12-8
-nologLink 12-8
GenSys Batch Mode Specific Options 12-8
-gen pdf <files> <destdir> 12-9
-gen sequence 12-9
-gen rtl 12-9
-usergen 12-9
-stop_untouched_hierarchy 12-9

13. Working with Registers

Capturing Component Register Information Using GUI 13-2
Specifying Register Definitions 13-2
Specifying Register Description 13-4
Specifying Reset Information 13-5
Modifying Register Details 13-5
Specifying Bitfield Definitions 13-5
Modifying or Deleting Bitfield Definitions 13-9
Copying Bitfield Definitions 13-10
Interpreting the Type of a Bitfield 13-11
Completing Bitfield Definitions 13-11
Viewing Register Map 13-12
Specifying Bit-Enums 13-14
Creating Register Groups 13-16
Capturing Component Register Information Using Tcl 13-17
Adding Registers 13-18
Adding Bitfields 13-18
Adding Bit-enums 13-18
Grouping Registers 13-19
Capturing Component Memory Information Using GUI 13-19
Generating Component-Level Register Map View 13-21
Levels of Functions 13-21
Register / Bitfield / Enum Names 13-22
Write-To-Decrement Registers 13-23
Creating Channels and Bridges 13-23

Creating Channels 13-23
Creating Bridges 13-25
Understanding Elaboration 13-27
Example of Elaboration in Case of Multiple Channels and Bridges 13-28
Handling Remap States During Elaboration 13-31

14. Working with Memory Maps

Understanding Memory Maps 14-2
Specifying MAU for a Requestor Interface 14-3
Component-Level Memory Maps 14-3
Design-Level Memory Map 14-6
Specifying Memory Type 14-8

Adding, Modifying, and Viewing Memory-Maps 14-9
Adding Memory Maps in the Tabular View 14-9
Adding Memory Maps in the Canvas View 14-9
Modifying Memory Maps 14-11
Adding Requestors and Targets in a Memory Map in the Canvas View . . . 14-12
Viewing Memory Maps with Assembly 14-13

Generating Memory Map View 14-13
Generating Component-Level Memory Map View 14-14
Generating Design-Level Memory Map View 14-15
Case1: Design Does Not Contain Any Subsystem (Simple Case) 14-15
Case2: Design Contains subsystems (Complex Case) 14-16

Viewing Graphical Representation of Register and Memory-Maps 14-19
Viewing Register Information 14-19
Viewing Memory-Map Information 14-20

Appendix A. The Group/Ungroup Feature

Functions and Characteristics of Group/Ungroup Feature A-2
Identifying the Intended Use Areas A-2
Use Model A-2
Full Logic Equivalence A-3
Effect on Connections Across the Subsystem Hierarchy A-3

Principles Governing the Grouping/Ungrouping Feature A-4
Principles Governing Grouping A-4
Principles Governing Ungrouping A-4

Scenarios of Grouping A-5
Single Adhoc/Interface Connections Scenario. A-5
Tieoff and Open Connections A-6
External Fanout Scenario A-7
Internal Fanout Scenario A-8
Scenarios of Ungrouping A-9
Ungrouping Scenario Within Adhoc and Outside Adhoc A-10
A More Complex Case of Adhoc-Adhoc A-11
Ungroup Scenario in Case of Adhoc and Interface/Spliced. A-12
Ungroup Scenario in Case of Interface/spliced and Interface A-13
Special case of Ungrouping Spliced and Spliced A-14
Ungrouping Scenario in Case of Tieoffs A-15
Ungroup Scenario in Case of Adhoc scalar and Tieoff A-15
Ungroup Scenario in Case of Adhoc Vector and Tieoff. A-16
Ungroup Scenario in Case of Interface/Spliced and Tieoff. A-17
Treatment to Attributes in Tieoff A-17
Temporary Tieoffs A-17
Ungroup Scenario in Case of Open. A-18
Ungroup Scenario in Case of Adhoc scalar and Open A-18
Ungroup Scenario in Case of Adhoc vector and Open. A-18
Ungroup Scenario in Case of Interface/Spliced and Open. A-19
Treatment to Attributes in Open A-19
Temporary Open. A-19
Ungroup Scenario in Case of Nets A-19
Ungroup Scenario in Case of Feedthroughs and Feedbacks A-19
Parameters and Instance Names During Ungrouping A-20
Treating Parameters During Ungrouping. A-20
Treating Instance Names During Ungrouping A-21

Appendix B. Using Configurable IPs

Configurable IP Definition B-2
Simple Configuration B-2
HDL Parameters. B-2
Complex Configuration B-2
Using Configurable IP. B-2
Compiler Directive Support. B-3
Example. B-3

Preface

This preface includes the following sections:

- About This User Guide
- Customer Support

About This User Guide

GenSys® provides an environment for SoC assembly and integration. It enables you to define, capture, and analyze high-level chip and subsystem architectures.

The GenSys User Guide describes GenSys concepts. It shows you how to use GenSys for:

- Design assembly and architectural analysis.
- Interactive architectural specification capture.
- Adding, removing, and connecting IPs.
- Validating connections to ensure that they are correctly hooked up.
- Automatic hierarchy manipulation to meet architectural physical specifications.

Related Publications

For additional information about GenSys, see the documentation on SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *GenSys Release Notes* on the SolvNet site.

To see the *GenSys Release Notes*,

1. Go to the SolvNet Download Center located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

2. Select GenSys, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*.
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Reporting an Error

The Technical Publications team welcomes your feedback and suggestions on this publication. Provide specific feedback and, if possible, attach a snapshot. Send your feedback at spyglass_support@synopsys.com.

Accessing SolvNet

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNet site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNet site, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at

    https://www.synopsys.com/support/global-support-centers.html
- Telephone your local support center.

- Call (800) 245-8005 from within North America.
- Find other local support center telephone numbers at https://www.synopsys.com/support/global-support-centers.html

1

Using the GenSys Tcl Commands

With an increase in design complexity coupled with reduced time-to-market, semiconductor product development companies have started to focus on IP reuse and platform-based design techniques with renewed vigor.

However, the integration of multiple IPs from different vendors in platform-based designs have resulted in SoCs becoming more complex with more pins, more I/O, mixed analog, digital technologies, and more power domains. This has resulted in complications in the current design flows.

Further, the need for higher design efficiencies in semiconductor product development has led to an increased focus on IP reuse and platform-based design techniques.

GenSys enables you to provide a resolution to the above concerns. It provides an environment for SoC assembly and integration. It enables you to define, capture, and analyze high-level chip and subsystem architectures.

It provides an environment for:

- Design assembly and architectural analysis.
- Interactive architectural specification capture.
- Adding, removing, and connecting IPs.
- Validating connections to ensure that they are correctly hooked up.
- Automatic hierarchy manipulation to meet architectural physical specifications.

GenSys Concepts

This section describes the following:

GenSys Objects	Formats for Saving GenSys Objects	GenSys Files
Using VLNV to Identify an Object	Stop Modules	

GenSys Objects

While working in GenSys, you basically work on the following types of objects:

Designs	Components	Interfaces	Design Configurations

Designs

A design refers to a chip or a subsystem.

It contains instances of various GenSys objects and the connections between them.

Components

A component represents a leaf-level IP. It defines the characteristics of an IP that is used to build Designs.

Typical components are CPU, DSP, UART, and USB interface.

Interfaces

An interface refers to a set of port definitions defining a standard (internal or external) bus/protocol. The examples of interfaces are PCI IO interface, USB IO interface, Timer IO interface, AXI, and AHB.

An interface definition typically contains a collection of logical ports. When you instantiate an interface on a component, you need to map the logical ports of the interface with the actual ports of the component. Once this is done, you can connect two component instances with the interfaces. For details, see Mapping Interfaces.

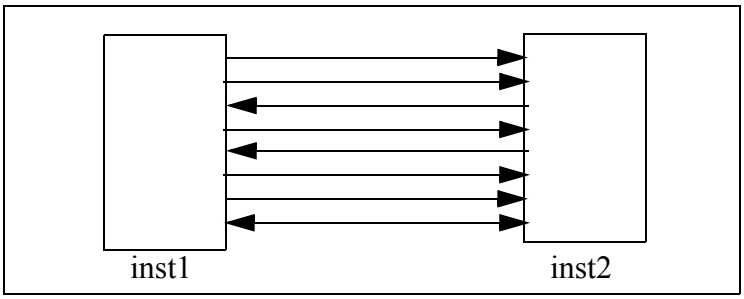
Advantages of Using Interfaces

The advantages of using interfaces are:

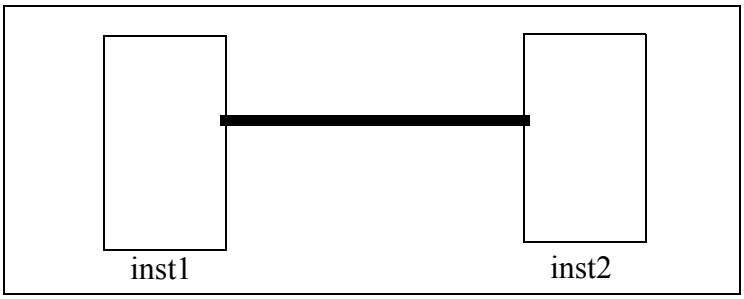
- They increase the level of abstraction and enable easy connectivity and design debug.
- They reduce the manual effort of creating huge number of connections.

  When you connect two interfaces, the ports or signals of the two interfaces get connected in one go. This saves a lot of time in manually connecting the ports or signals, especially if the port or signals number is huge (say in thousands).

  For example, consider the connections between two instances in the following figure:



In the above scenario, one interface can replace all the above connections, as shown in the following figure:



Design Configurations

A design configuration is an XML file that enables you to specify configuration details for various instances in a design.

You can create this file similar to Designs, Components, or Interfaces.

Formats for Saving GenSys Objects

You can save objects in any of the following formats:

- GenSys XML
  - It is the default format (MRS).
  - It is the super set of IP-XACT.
  - It is ideal for work-in-progress designs.
  - You can export this format to IP-XACT while interfacing with other tools. In this case, GenSys saves the extra information as Atrenta vendor extensions.
- IP-XACT XML
  - It is an IEEE standard.
  - It is supported by several IP and tool vendors.
  - Gensys supports the IP-XACT version 1.2, 1.4, and IEEE 1685-2009.
- Tcl Files

  Use this format if you want to recreate designs in the batch mode by using Makefile.

Supported Formats During Object Import and Export

GenSys supports the following formats during import and export of objects:

- GenSys XML
- IP-XACT
- Tcl
- RTL
- MS Excel

Using VLNV to Identify an Object

VLVN stands for Vendor Library Version Name. These four parameters enables to identify an object uniquely.

For example, two objects of different versions are considered as separate entities even if they have the same name, library, and vendor.

Stop Modules

Stop modules are the sub-systems beyond which RTL import does not occur. Such sub-systems:

- Appear as black boxes under the Design Browser Tab.
- Are passed to SpyGlass with the -stop command.

  Sub-systems specified with the -stop command in SpyGlass are not considered for SpyGlass analysis.

For information on RTL import and configuring sub-systems as stop modules, see Chapter 4, “Performing RTL Import and Configuring Stop Modules and Macros.

GenSys Files

GenSys provides the following files:

- Configuration File
- The gensysrc File

Configuration File

The configuration file, .gensys.setup, is an ASCII text file that contains configuration settings, such library files path, output directory, path of Verilog and VHDL directories, report directories, and path of log files.

This file contains configuration information in the form of Tcl commands and procedures.

The file can also have comments starting with #. In addition, all user-defined environment variables are supported in the configuration file.

Reading the Configuration File

GenSys reads .gensys.setup from the following locations in the given sequence and uses the first found file:

- The configuration file present in the current working directory
- The configuration file present in the home directory ($HOME)
- The configuration file present in the directory specified by the GENSYS_SETUP environment variable.

Commonly Used Commands in a Configuration File

The following table describes the commonly used commands in a configuration file:

Command	Description
set_library_path	Sets a library path
set_logs_dir	Sets a path for log files
set_reports_dir	Sets a path for report files
set_verilog_dir	Sets a path for the generated Verilog files
set_vhdl_dir	Sets the path for the generated VHDL files
set_default_save_dir	Sets a default path to save the design objects.
set_default_save_format	Sets a default save format for the design objects being saved. For details on save formats, see Formats for Saving GenSys Objects.

Sample Configuration File

The sample configuration file is as follows:

```
set_verilog_dir ./Verilog
set_log_dir ./Logs
set_report_dir ./Reports
set_default_save_format MRS
set_default_save_dir ./Library/local
set_save_template mrs "mrs_dir/%vendor/%library/%version/%obj.xml"
set_save_template ipxact "ipxact_dir/%vendor/%library/%version/%obj.xml"
set_save_template tcl "tcl_dir/%vendor/%library/%version/%obj.tcl"
set_rtlimport_option -load_from_library_browser FALSE
set_library_path -a -r Library/local/mrs_dir Library/local/ipxact_dir ./
Library/reference
```

The gensysrc File

GenSys stores its default settings in the ~/.gensys/gensysrc file. These settings include the details, such as the default background color, pin color, or plane color of the schematic window, visibility setting for various tabs and tables, the default font settings, etc. When you invoke GenSys, these default settings are picked from the ~/.gensys/gensysrc file.

Reading the gensysrc File

GenSys reads the gensysrc file from the following locations in the following order and uses the first found file:

- Installation directory, SPYGLASS_HOME
- Project home directory set using the $GENSYS_PROJECT_HOME environment variable
- User home directory

Impact on the file on the gensysrc file while updating GenSys settings

While working in GenSys, you may change some settings, such as hiding the visibility of some tables or changing the plane color in schematic. In such a case, GenSys saves the changes in the gensysrc file stored only under the user home directory. The other gensysrc files, if present at the other two locations, remain unchanged. While writing the changes back to the gensysrc file of the user's home directory, GenSys uses the following assertions:

- Any setting which was not read and has been changed is written in the gensysrc file. The other unchanged entries are not written.
- All entries which were read from the gensysrc file are written back even if they have not been changed.

Getting Started with GenSys

Before you invoke GenSys, perform the following actions:

- Setting the GenSys Path
- Setting the GenSys License
- Setting the Configuration File
- Setting the Directory Structure

Setting the GenSys Path

Set the GENSYS_HOME environment variable to point to the installation path of GenSys.

csh

```
%>setenv GENSYS_INSTALL_PATH <full-path-name-of-release> %>setenv
GENSYS_HOME $GENSYS_INSTALL_PATH/SPYGLASS_HOME
```

sh, bash

```
%> export GENSYS_HOME <full-path-name-of-release>/SPYGLASS_HOME
```

Place GenSys in your path by using the following command:

```
%> set path = ($GENSYS_HOME/bin $path)
```

Setting the GenSys License

Set the complete path name of the license file by using the following commands:

csh

```
%> setenv ATRENTA_LICENSE_FILE <full-path-name-of-license-file>
```

sh, bash

```
%> export ATRENTA_LICENSE_FILE <full-path-name-of-license-file>
```

If the license file is already available on the license server, ensure that the license server and the corresponding port is part of the LM_LICENSE_FILE setup.

Alternatively, you can also source the setup file located in the <complete-workshop-install-dir-path>/Gensys_setup.csh file by using the following command:

```
%> cd <gensys-install-dir-path> %> source GenSys_setup.csh
```

Setting the Configuration File

Configure GenSys settings in the configuration file gensys.setup.

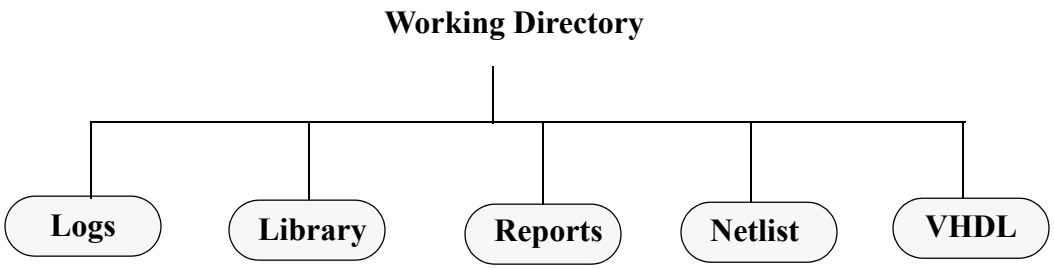
For details on this file, see Configuration File.

Setting the Directory Structure

A proper directory structure provides an organized work area.

You can create a directory structure by specifying Tcl commands either in the Configuration File or the *Command* text box in the Command/Log Pane.

For example, you can have the following directory structure to store different types of files:



This section contains the following topics:

Setting the Library Path
Setting the Output Directory
Setting the Verilog and VHDL Directories
Setting the Report Directory
Setting the Log Directory

Setting the Library Path

The library path specifies the path from where GenSys should pick libraries by default. GenSys loads these libraries under the Library Browser tab.

Once you set the library path, you do not need to explicitly load libraries from the specified path by using RTL import every time you invoke GenSys.

To specify a library path, use the set_library_path Tcl command. For details on this command, refer to *GenSys Tcl Commands Reference Guide*.

Setting the Output Directory

By default, the output directory for GenSys output is the current working directory (from where GenSys has been invoked.)

You can set your own output directory in any of the following ways:

- By using the -outdir command-line option while invoking GenSys.
- By using the File > Set Output Directory menu option in GenSys GUI.
- By using the set_output_dir Tcl command.

Setting the Verilog and VHDL Directories

When you generate RTL, GenSys saves the generated Verilog and VHDL files in the Verilog and VHDL directory, respectively.

The following preference is used (from the highest priority) to create a default directory for storing the generated VHDL/Verilog files:

1. Directory specified by the set_vhdl_dir/set_verilog_dir Tcl command.
2. Netlist directory specified by the File > Set Netlist Directory menu option or the set_netlist_dir Tcl command (if the set_vhdl_dir/set_verilog_dir Tcl commands are not specified).
3. The output/VHDL and output/Verilog directories.

   Note:
   In case of partitions, GenSys creates the *<default-dir>/<partition-name>* sub directories.

Setting the Report Directory

By default, GenSys saves all reports in the report directory.

To store the reports in another directory, specify the directory in any of the following ways:

- By using the File > Set Reports Directory menu option
- By using the set_report_dir Tcl command

Setting the Log Directory

By default, GenSys stores all the log files in the log directory under the current directory.

To specify a different directory to store the log files, use the set_log_dir Tcl command. You can specify this command in the configuration file also.

Directories Created Under the Log Directory

Under the log directory specified by the set_log_dir Tcl command, GenSys creates subdirectories for each run in a GenSys session.

For example, if you specify the directory as myLog by using the set_log_dir Tcl command and then perform three runs in a GenSys session, the following directories are creating storing the log details of each session:

myLog/Run_1

myLog/Run_2

myLog/Run_3

When you close the current session and start a new session, all the Run_* directories of the previous session are deleted and created afresh based on the runs in the current GenSys session.

Directory to Store Internally-Generated Files

All internally-generated files, such as SpyGlass® log files, SDI log files, and other internal output are saved in the .gensys_dir directory structure.

Within this directory structure, the tmp sub-directory stores all SpyGlass log files and SDI log files. These sub-directories further have appropriate sub-directories with the name of the generator, such as SpyGlass, SDI, RMEs, and CDC, which led to the generation of these internal files.

Invoking GenSys

You can invoke GenSys either in the GUI or the batch mode.

Invoking GenSys in the GUI Mode

This is the default mode of GenSys.

To invoke GenSys in the GUI mode, perform the following steps:

1. Set the GENSYS_HOME environment variable to the SPYGLASS_HOME directory.
2. Launch GenSys GUI by using the $GENSYS_HOME/bin/gensys -product <*product*> command.

   If you do not specify any product, GenSys invokes the Assembly product.

In this mode, you can choose between The Canvas View (recommended) or The Tabular View.

Invoking GenSys in the Batch Mode

In the batch mode, you use command-line options to perform various tasks. For details on these command-line options, see Using GenSys in the Batch Mode.

To invoke GenSys in the batch mode, specify the -batch command-line option while invoking GenSys.

Displaying Debug Messages

Use the -debug option with the gensys command to enable the display of debug/processing messages in the Message Log window of GenSys GUI. If you do not use the -debug option, GenSys displays only errors or warning messages in the GUI. Debug/processing messages are not shown in the Message Log window.

Introducing GenSys GUI

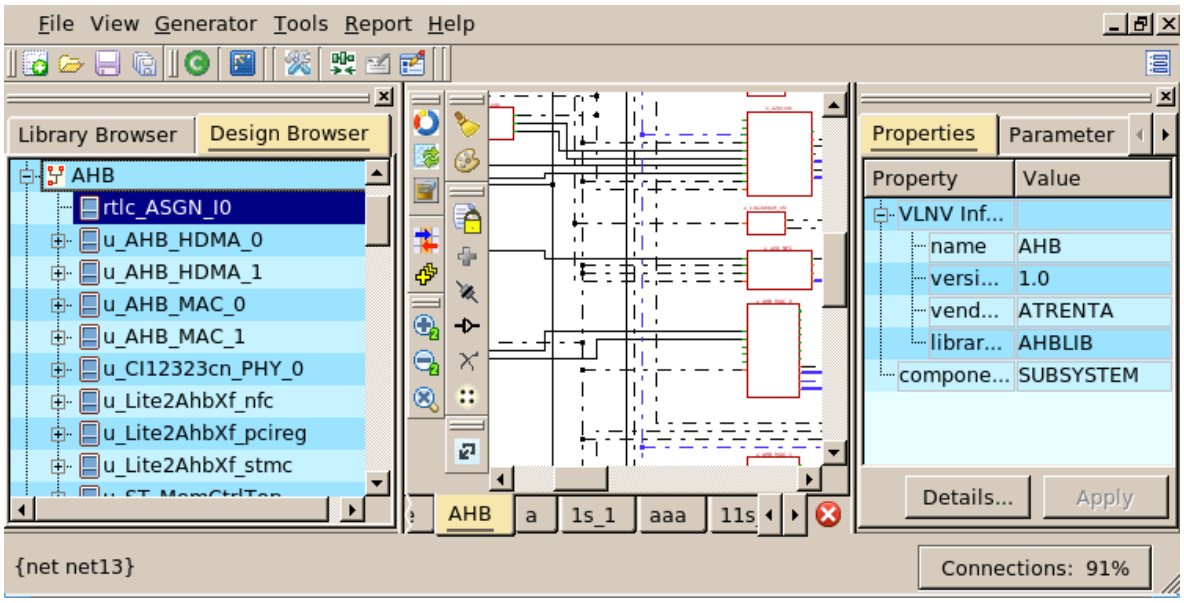
You can use GenSys GUI in any of the following views:

- The Canvas View
- The Tabular View

The Canvas View

In this view, you work directly in the schematic of the currently loaded object.

The following figure shows GenSys GUI in the tabular view:



To switch to the canvas view, click

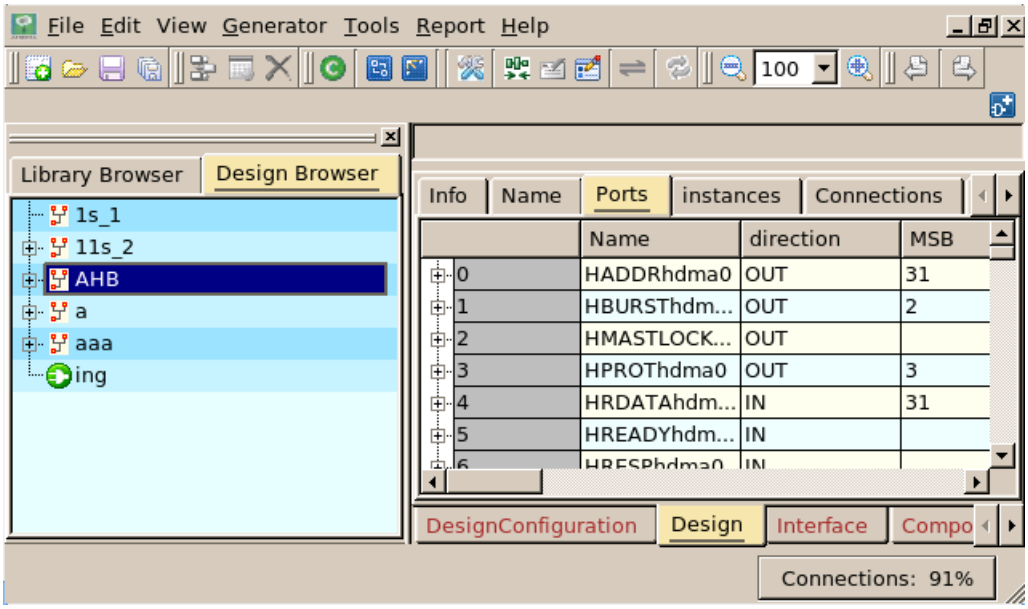
button in the GUI window.

For details on this view, see Using the Canvas View.

The Tabular View

In this view, you work in a spreadsheet-like structure.

The following figure shows GenSys GUI in the tabular view:



To switch to the tabular view, click

button in the GUI window.

For details on this view, see Using the Tabular View.

Work Area in the GUI Mode

In the GUI mode, the GenSys work area is organized into sections, such as menu bar, toolbar, windows, and panes, as described below:

- The Design Browser Tab that displays currently loaded components, interfaces, or designs.
- The Library Browser Tab that displays all the previously loaded components, interfaces, and design. It also displays currently loaded design, components, or interfaces.
- The IP-XACT Viewer Pane displays the details of the underlying IP-XACT (XML) of the selected object
- The Command/Log Pane that displays message information generated by GenSys.
- The Modular Schematic Window that displays schematic of the selected design or component.
- The Incremental Schematic Window that displays the selected portions of a (flattened) design schematic across hierarchical boundaries.
- The Statistics Window that shows connection summary of a design in a graphical format.
- The Menu Bar that contains GenSys menu options.
- The Toolbar that contains icons for important functions.

Closing GenSys

To exit GenSys, select the *File -> Exit* menu option.

If there are any unsaved changes, GenSys prompts you to save the changes.

GenSys Exit Status

GenSys generates an exit status code that provides the exact status of the GenSys run.

Exit Code	Description
0	GenSys run was successful, i.e. no error or warning messages.

Exit Code	Description
11	GenSys run was completed without any error messages but has warning messages. The generated output is correct, but you should review the warning messages.
12	GenSys run was completed with error messages. The generated output may not be correct and you should review the warning messages.
13	GenSys run was completed with error messages due to Tcl built-in or unknown commands. The generated output may not be correct and you should review the warning messages.
1	GenSys run was terminated due to an abnormal error that is not trapped by GenSys In this case of termination due to an abnormal error, the gensys_crash.log file, which contains the stack trace, is dumped in the log directory. In addition, the information describing the reason of crash, which are details of signals that caused the error are also appended to the gensys.log file. These signal details can be SIGINT, SIGQUIT, SIGTERM, SIGXCPU, SIGPIPE, SIGSEGV, SIGFPE, SIGABRT, and SIGILL.
3	GenSys run was terminated due to an error that is trapped by GenSys. Review the log.
4	GenSys run was terminated due to a license failure. Check your license.
5	GenSys run was terminated forcibly.

2

Introducing GenSys Flows

Based on your requirement, you can choose between any of the following flows while working in GenSys:

- The IP Packaging Flow
- The RTL Restructuring Flow
- The Structured Assembly Flow

The IP Packaging Flow

Use this flow to package the RTL of an IP in a reusable IP-XACT format.

During IP-packaging, GenSys extracts the following information from the RTL of an IP to create a packaged view of the IP:

- Ports and parameter information.
- Logical library information, use clauses, package and include files, and the pre-processor macros that define an IP.

IP packaging is not only limited to extracting RTL view of an IP, but also involves the following:

- Creating bus interfaces and mapping them to RTL ports
- Annotating properties, such as clocks, reset, and data to ports
- Adding register and memory map information
- Adding additional software or simulation views to the IP

Using IP-XACT in GenSys

IP-XACT is an XML format based on the open standard defined by the SPIRIT consortium. The IP-XACT specification is used for maintaining the IP information by using meta data and design exchange format.

The IP-XACT standard provides a set of Semantic Consistency Rules (SCR) to validate the IP-XACT database.
GenSys supports the IP-XACT version 1.2, 1.4, and IEEE 1685-2009 standard to describe and configure IPs for efficient integration of IPs in SoC designs.

Note:
Currently, not all IP-XACT constructs are supported through the GenSys GUI or Tcl commands. For unsupported constructs, you can edit the XML separately.
Wired assembly constructs are completely supported from within GenSys. All others, including Registers, TLM, and Generator Chains, are partially supported from within GenSys.

Loading the IP-XACT Data

To load the IP-XACT data in GenSys, perform the following steps:

1. Select the File > Open menu option. This displays the Select one or more files to open dialog.

2. Select the IPXACT XML Files (*.xml) option from the File Type drop-down list.
3. Select the required file.
4. Click the Open button.

Alternatively, use the load Tcl command to load the IP-XACT data, as shown in the following example:

```
load design1.xml -as IPXACT
```

IP-XACT VLNV matching is case-insensitive. For example, consider an interface instantiated in a component with the following VLNV information:

Vendor	AMBA
Library	Busdef
Name	AHB
Version	r0p0

Also consider another interface, InterfaceLib, present in the library with the following VLNV information:

Vendor	amba
Library	busdef
Name	ahb
Version	r0p0

In this case, the two are treated as same, and InterfaceLib is loaded from library and is instantiated in the component.

Translation of IP-XACT Files

GenSys automatically translates IP-XACT XML files to populate GenSys internal data structure (COM). This translation also synchronizes GenSys COM with the IP-XACT data structure. The translation is performed when you open/load an IP-XACT XML file in GenSys GUI and is carried out to generate RTL from the IP-XACT design files.

Currently, RTL generation is performed only for the connections, interfaces, ports, and parameters in the design.

Specifying the Default Version of IP-XACT Files

You can set the default version for IP-XACT files by using the set_ipxact_version Tcl command. By default, GenSys saves the objects in IP-XACT version 1.4. You can change the default IP-XACT version to 1.2 by using the set_ipxact_version Tcl command, as shown below:

```
set_ipxact_version 1.2
```

Saving the IP-XACT Data

Use the save Tcl command to save the IP-XACT data.

While saving the IP-XACT data, if you do not specify the version, vendor, and library details, GenSys uses the following default values:

- GENESIS_DEFAULT_VERSION
- GENESIS_DEFAULT_VENDOR
- GENESIS_DEFAULT_LIBRARY

While saving the designs loaded from IP-XACT XML files, for the adhoc and saved adhoc connections, the hyper edges (with more than two pin references) are resolved into different pin-to-pin adhoc connections.

Note:
While saving the IP-XACT files in TCL or MRS format, GenSys also dumps the IP-XACT extension tables.

Validating the IP-XACT Files

When you load the IP-XACT XML files, GenSys performs validation of the XML files to ensure that the XML files are adhering to the schema definition provided in the XSD (XML Schema Definition) files.

You enable or disable such validation, use the validate_ipxactxml Tcl command. For details on this Tcl command, refer to *GenSys Tcl Commands Reference Guide*.

Using Direction and Bit-width Constraints

GenSys provides support of IP-XACT 1.4 direction and bit-width constraints. These constraints are used for restricting logical port direction and bit-width in an actual interface instantiation.

Note:
You cannot specify these constraints in GenSys. You can use these constraints only if they are present in the imported RTL.

These constraints are different for logical ports of a master interface, a slave interface, and a system interface. In addition, these constraints are also different for different group attributes of a system interface.

Therefore, bit-width and direction constraint checks for an interface library instantiated as a master will be different from constraint checks for an interface library instantiated as a slave.

Using the Direction Constraints

A direction constraint specifies the direction of a logical port. Based on the direction, the logical port can be mapped to a physical port having an identical direction.

If the direction of a physical port is different from the direction of a logical port being mapped, GenSys reports an error.
For example, consider a scenario in which the OnMaster constraint direction for a logical port, L1, of intf interface is in and OnSlave constraint direction for that port is out. In this case:

- If intf is instantiated as a master, the physical (actual) port mapped to L1 must have in direction.
- If intf is instantiated as a slave, the physical port mapped to L1 must have out direction.

Direction Constraint not Present

If no direction constraint is present for a logical port of an interface (master, slave or system), that logical port will have the same direction as actual port and can be mapped to physical port of any direction.

Using Bit-Width Constraints

The bit-width constraint specifies the maximum bit-width of a logical to physical port map for an interface instantiation (master, slave, or system).

Consider a scenario in which the bit-width constraint on logical port L1 for master is 5 and for slave is 3. In this case, logical to physical port map for L1 cannot have a width greater than 5 in master interface.
So assuming P1 as the actual physical port mapped to L1, port maps L1 (4:0)-P1 (4:0) and L1(6:2)-P1(4:0) are correct in this case. However, if the port map is L1(7:1)- P1(6:0), GenSys reports an error.

Similarly for slave interface, port map L1(2:0)- P1(4:2) is correct, but port map L1(5:0) - P1(5:0) is not correct.

Bit-Width Constraint not Present

If no bit-width constraint is specified, logical port-mapping will have the same width as physical port-mapping

Mapping Single Logical Port to Multiple Actual Ports

GenSys provides IP-XACT 1.4 support for mapping a single logical port to multiple actual ports. This is done by mapping different slices of the same logical port to different actual ports. For example L1(2:0) can be mapped to A1(2:0) and L(5:3) can be mapped to A2(2:0).

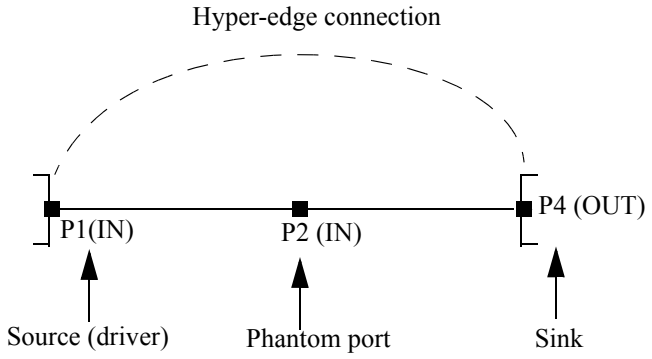
Note:

This support is currently provided for IP-XACT load and save as well as in GenSys netlisting and elaboration. However, this support is not provided in MRS, Tcl read and write, and in GenSys GUI view. In addition, splice interface connection and splice tieoff connections are not supported for this feature.

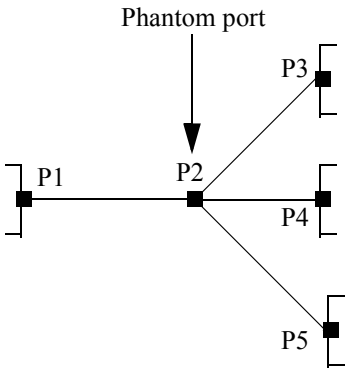
Using Phantom Ports

GenSys provides IP-XACT 1.4 support for hyper-edge interface connection through phantom ports.

A phantom port is a dummy physical port of direction phantom. Such ports are used to create hyper-edge interface connections.
A hyper-edge connection is a connection in which a single source or driver is connected to multiple sinks.
The following figure shows a simple example of a hyper-edge connection between P1 and P4 ports through a phantom port P2:



In a hyper-edge connection, the direction of source and phantom port or the direction of sink and phantom port can be same. However, as the actual connection is between the source and sink, the direction of these ports should be different.
You can map multiple logical ports to the same phantom port. An example is shown in the following figure:



Note:
Phantom ports are not dumped in the generated RTL.

Using Monitor Interfaces

GenSys provides IP-XACT 1.4 support for monitor interfaces. A monitor interface is mainly used for scanning purpose, and therefore, is beyond the scope of any netlist error checking.

Please note the following points:

- GenSys does not perform any netlist error-checking and elaboration connectivity error checks for monitor interfaces.
- Export interface operation does not work on monitor interface.
- Monitor interface connection can also have hyper-edge connection. GenSys netlisting detects the correct source for that connection graph and connect it to all the sinks of the hyper-edge.

Using ALDA Ports

GenSys provides IP-XACT 1.4 support of All Logical Directions Allowed (ALDA) ports. These ports are similar to phantom ports but are applicable at pin-level.

If you set this attribute to true for a physical port, a logical port of any direction can be mapped to this physical port.
Please note the following points:

- ALDA ports are dumped in the generated RTL.
- ALDA ports are not applicable for phantom ports and inout ports. They are applicable only for input or output ports.

- ALDA port connections can have hyper-edge connections. Therefore, connection priority checks and other connectivity checks are not performed for ALDA ports. Finally netlisting detects the driver (source) for this hyper-edge and connects it to all the sinks.

  Note:
  Support hyper edge is given in netlisting only, other GenSys generators like RTLHealthCheck and Schematic generator still do not understand hyper-edge and hence they will not work correctly for phantom ports, monitor interface and ALDA.

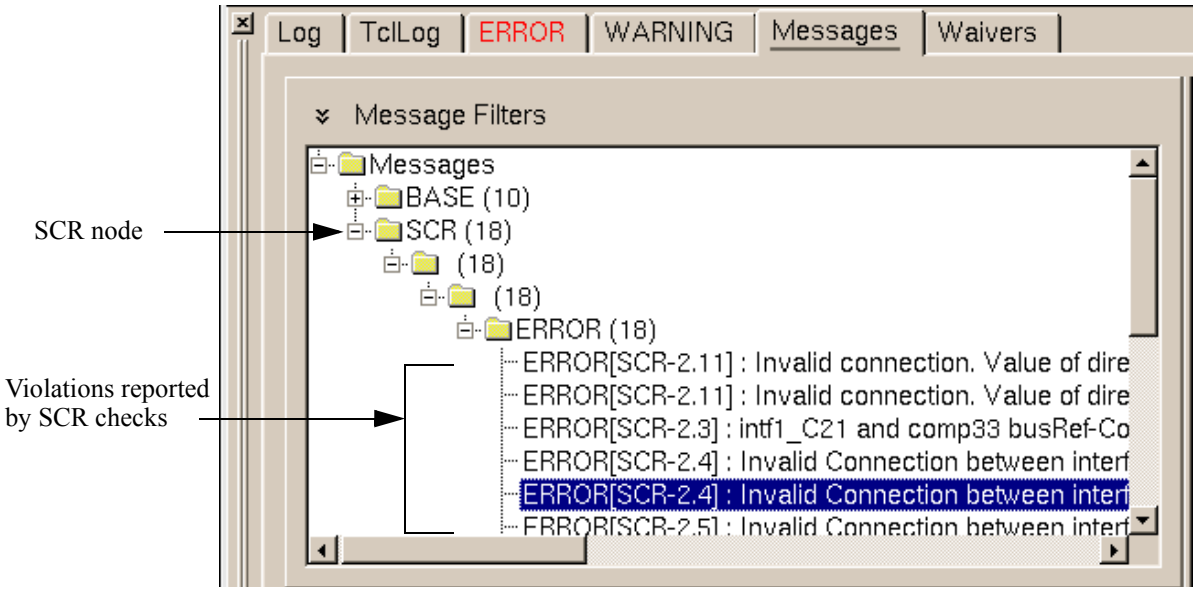
Validating the IP-XACT Data by Running the SCR Checks

The Semantic Consistency Rules (SCR) are the assembly and registers related checks defined by the IP-XACT standard. These checks validate the correctness of IP-XACT files.

You can run these checks hierarchical or flat.

Viewing Violations of SCR Checks

You can view violations reported by these checks under the SCR node in the Messages tab of the Command/Log Pane. The following figure shows an example of SCR checks reported under the Messages tab:



Enabling the SCR Checks

Use the following Tcl command to enable SCR checks while loading IP-XACT files:

```
set_gensys_options -enable_scr_checks Y
```

Note:

The SCR checks are always enabled while saving IP-XACT files. This ensures that the clean IP-XACT database is being saved.

Running the SCR Checks

Use the run_scr_checks Tcl command to run SCR checks.

When you run this command, GenSys displays the SCR violations with the SCR number as per the IP-XACT standard and the SCR description in the Command/Log Pane, as shown in the following figure:

```
[COMMAND] : run_scr_checks
ERROR[SCR-2.3] : busRef : 'intf1_C21' and compRef : 'comp33' busRef-compRef pair for interconnection named 'ic4'
ERROR[SCR-2.4] : Interconnection named 'ic2' between interface 'intf1_C20' (component instance : 'comp11') and int
ERROR[SCR-2.4] : Interconnection named 'ic4' between interface 'intf2_C20' (component instance : 'comp11') and int
ERROR[SCR-2.5] : Interconnection named 'ic2' between interface 'intf1_C20' (component instance : 'comp11') and int
ERROR[SCR-2.5] : Interconnection named 'ic4' between interface 'intf2_C20' (component instance : 'comp11') and int
ERROR[SCR-2.6] : Interconnection named 'ic3' between interface 'intf2_C21' (component instance : 'comp11') and int
ERROR[SCR-2.7] : Interconnection named 'ic3' between interface 'intf2_C21' (component instance : 'comp11') and int
ERROR[SCR-4.1] : Active interface 'intf1_C20' (component instance : 'comp_monitor') in design 'dsgn1' does not refe
ERROR[SCR-4.2] : Monitor interface 'intf1_C21' (component instance : 'comp_monitor') used in the interconnection r
ERROR[SCR-4.2] : Monitor interface 'intf1_C20' (component instance : 'comp_master') used in the interconnection na
ERROR[SCR-4.3] : The value of the interfaceMode : 'slave' of the monitor interface 'intf1_C20' (component instance :
```

GenSys also saves the above violations in the pbd.log file.

Modifying SCR Checks

You can modify the SCR checks by changing their severities. You can also enable or disable some SCR checks.

To do so, use the *Rules Setup* dialog. For details, see Setup Rules.

Supported SCR Checks in GenSys

The following tables state the support provided by GenSys for each SCR rule.

Cross-References and VLNVs

Table 2-1

SCR Rule	Supported	Product	Comments
SCR 1.1	Yes	Assembly	LIBRARY_SCAN
SCR 1.2	Yes	Assembly	LIBRARY_SCAN
SCR 1.3	Yes	Assembly	LIBRARY_SCAN
SCR 1.4	Yes	Assembly	LIBRARY_SCAN
SCR 1.5	Yes	Assembly	LIBRARY_SCAN
SCR 1.6	No	Assembly	generatorChain related
SCR 1.7	No	Assembly	generatorChain related
SCR 1.8	Yes	Assembly	LIBRARY_SCAN
SCR 1.9	Yes	Assembly	LIBRARY_SCAN
SCR 1.10	Yes	Assembly	LIBRARY_SCAN
SCR 1.11	Yes	Assembly	LIBRARY_SCAN
SCR 1.12	No	Assembly	abstractor related
SCR 1.13	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

Interconnections

SCR Rule	Supported	Product	Comments
SCR 2.1	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.2	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.3	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

SCR Rule	Supported	Product	Comments
SCR 2.4	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.5	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.6	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.7	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.8	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.9	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.10	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.11	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.12	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.13	Yes	Assembly	IPXACT_LOAD
SCR 2.14	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 2.15	Yes	Assembly	IPXACT_LOAD

Channels, Bridges, and Abstractors

SCR Rule	Supported	Product	Comments
SCR 3.1	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 3.2	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 3.3	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 3.4	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 3.5	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 3.6	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 3.7	No	Assembly	related to abstractor
SCR 3.8	No	Assembly	related to abstractor
SCR 3.9	No	Assembly	related to abstractor
SCR 3.10	No	Assembly	related to abstractor
SCR 3.11	No	Assembly	related to abstractor
SCR 3.12	No	Assembly	related to abstractor
SCR 3.13	No	Assembly	related to abstractor
SCR 3.14	No	Assembly	related to abstractor
SCR 3.15	No	Assembly	related to abstractor
SCR 3.16	No	Assembly	related to abstractor
SCR 3.17	No	Assembly	related to abstractor
SCR 3.18	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

Monitor Interfaces and Monitor Interconnections

SCR Rule	Supported	Product	Comments
SCR 4.1	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 4.2	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 4.3	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 4.4	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 4.5	Yes	Registers	ON_DEMAND(IPXACT_SAVE)

Configurable Elements

SCR Rule	Supported	Product	Comments
SCR 5.1	Yes	Assembly	IPXACT_LOAD
SCR 5.2	Yes	Assembly	IPXACT_LOAD
SCR 5.3	Yes	Assembly	IPXACT_LOAD
SCR 5.4	Yes	Assembly	IPXACT_LOAD
SCR 5.5	Yes	Assembly	IPXACT_LOAD
SCR 5.6	Yes	Assembly	IPXACT_LOAD
SCR 5.7	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 5.8	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 5.9	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 5.10	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

SCR Rule	Supported	Product	Comments
SCR 5.11	Yes	Assembly	IPXACT_LOAD
SCR 5.12	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 5.13	No	Assembly	generatorChain related
SCR 5.14	No	Assembly	generatorChain related
SCR 5.15	No	Assembly	related to abstractor

Ports

SCR Rule	Supported	Product	Comments
SCR 6.1	Yes	Assembly	IPXACT_LOAD
SCR 6.2	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.3	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.4	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.5	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.6	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.7	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.8	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.9	Yes	Assembly	IPXACT_LOAD
SCR 6.10	Yes	Assembly	IPXACT_LOAD
SCR 6.11	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.12	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.13	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.14	No	Assembly	related to abstractor
SCR 6.15	Yes	Assembly	IPXACT_LOAD
SCR 6.16	Yes	Assembly	IPXACT_LOAD
SCR 6.17	Yes	Assembly	IPXACT_LOAD

SCR Rule	Supported	Product	Comments
SCR 6.18	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.19	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.20	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.21	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.22	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.23	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.24	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 6.25	Yes	Assembly	IPXACT_LOAD
SCR 6.26	Yes	Assembly	IPXACT_LOAD
SCR 6.27	Yes	Assembly	IPXACT_LOAD

Registers

SCR Rule	Supported	Product	Comments
SCR 7.1	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 7.2	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 7.3	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 7.4	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 7.5	Yes	Registers	ON_DEMAND(IPXACT_SAVE)

Memory Maps

SCR Rule	Supported	Product	Comments
SCR 8.1	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 8.2	No	Registers	Related to banks
SCR 8.3	No	Registers	Related to banks
SCR 8.4	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 8.5	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 8.6	No	Registers	Related to banks
SCR 8.7	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 8.8	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 8.9	Yes	Registers	ON_DEMAND(IPXACT_SAVE)

Addressing

SCR Rule	Supported	Product	Comments
SCR 9.1	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 9.2	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 9.3	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 9.4	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 9.5	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 9.6	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 9.7	Yes	Registers	ON_DEMAND(IPXACT_SAVE)

Hierarchy

SCR Rule	Supported	Product	Comments
SCR 10.1	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.2	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.3	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.4	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.5	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.6	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

SCR Rule	Supported	Product	Comments
SCR 10.7	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.8	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.9	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.10	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.11	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 10.12	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

Hierarchy and Memory Maps

SCR Rule	Supported	Product	Comments
SCR 11.1	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 11.2	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 11.3	Yes	Registers	ON_DEMAND(IPXACT_SAVE)
SCR 11.4	Yes	Registers	ON_DEMAND(IPXACT_SAVE)

Constraints

SCR Rule	Supported	Product	Comments
SCR 12.1	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.2	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.3	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.4	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.5	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.6	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.7	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.8	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.9	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

SCR Rule	Supported	Product	Comments
SCR 12.10	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.11	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.12	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 12.13	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

Design Configurations

SCR Rule	Supported	Product	Comments
SCR 13.1	No	Assembly	generatorChain related
SCR 13.2	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 13.3	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 13.4	No	Assembly	abstractor related
SCR 13.5	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)
SCR 13.6	No	Assembly	abstractor related
SCR 13.7	No	Assembly	generatorChain related
SCR 13.8	No	Assembly	generatorChain related

Rules that Require External Knowledge

SCR Rule	Supported	Product	Comments
SCR 14.1	No	Assembly	external linkage related
SCR 14.2	No	Assembly	external linkage related
SCR 14.3	Yes	Assembly	ON_DEMAND(IPXACT_SAVE)

Defining the Scope of SCR Checks

Set the -hierarchy argument of the run_scr_checks Tcl command to true to run SCR checks on the entire design hierarchy including component and interface instances.

Set this argument to false to run SCR checks only on the active object (excluding the design hierarchy).

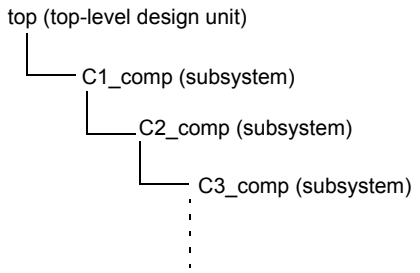
Viewing the IP-XACT Data in the IP-XACT Viewer Pane

For details, see IP-XACT Viewer Pane.

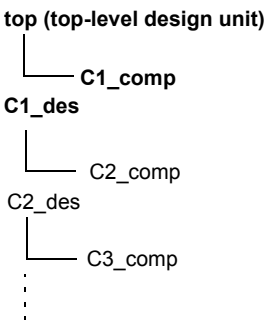
Additional Information on the IP-XACT Support

For IP-XACT support provided by GenSys, following information needs to be considered:

- In GenSys, if a component has a hierarchical reference of a design, that component is instantiated as a sub-system (design) in the top-level design unit. Therefore, GenSys displays one single view (design view) for the component having the design hierarchical reference. However, in IP-XACT, a component having a hierarchical reference of a design will have two separate views: one for the component (containing ports and memory description) and one for the design (containing instance and connection description). This concept is graphically explained in the following figure:

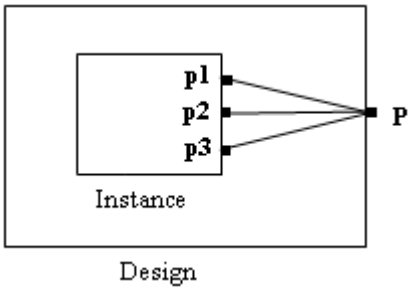


Top-level Design Hierarchy in GenSys



Top-level Design Hierarchy in IP-XACT

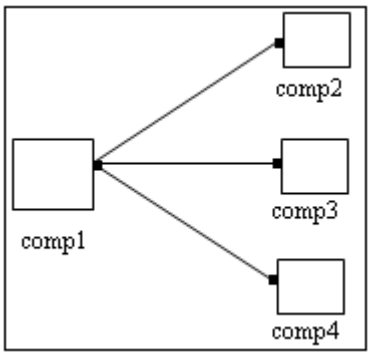
- For designs loaded from IP-XACT XML files:
  - If a saved adhoc connection contains reference of more than one instance pins then all such instance pins are saved to the same design port. An example for this is shown in the following figure:



While saving an adhoc connection, the direction of the design port is decided on the basis of the following criteria:

- If the direction of all the instance pins is IN then the direction of the design port is considered as IN.
- If the direction of all the instance pins is OUT then the direction of the design port is considered as OUT.
- If the direction of instance pins is a mixture of IN and OUT then the direction of the design port is considered as INOUT.
- If the design port of the same name already exists then the direction of that port is modified based on the above criteria.

- If an adhoc connection contains more than two pin references then a driver is selected among those pins. All other pins are connected to the driver. This is shown in the following figure:



The driver is selected based on the following criteria:

- If an OUT port is present then that port is considered as the driver.
- If an OUT port is not present but an INOUT port is present then that INOUT port is considered as the driver.
- If neither OUT nor INOUT port is present (i.e., all the ports are IN ports) the first specified pin is considered as the driver.

- If no reset value is specified for a register in IP-XACT XML file, a default reset value, 0, is considered for that register.
- The Logical Ports table in the Interface View of GenSys GUI has been enhanced to include two new fields, namely requiresDriver and driverType. The requiresDriver field specifies whether a signal in the interface requires a driver. The default value of this field is false.

The driverType specifies the type of the driver used for the signal. This field takes three values, *any*, *clock*, *singleShot*.

- Any logical port defined in GenSys will now have the following default attributes:
  - ON_MASTER_BITWIDTH
  - ON_MASTER_DIR
  - ON_SLAVE_BITWIDTH
  - ON_SLAVE_DIR
  - ON_SYSTEM_BITWIDTH
  - ON_SYSTEM_DIR
  - ON_SYSTEM_GROUP

  These attributes correspond to wiretypedef constraints section in abstractiondefintion. For more information on the wiretypedef elements, refer to Section 6.11.5.2 in the IP-XACT 1.4 Standard.

  The ON_MASTER_BITWIDTH and ON_MASTER_DIR attributes define the constraints for the signal when present in a master bus interface.

  The ON_SYSTEM_BITWIDTH, ON_SYSTEM_DIR, and ON_SYSTEM_GROUP attributes define the constraints for the signal when present in a system bus interface with a matching group name.

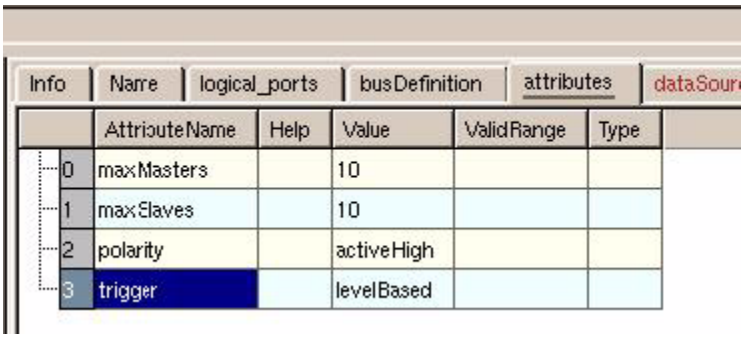
  The ON_SLAVE_BITWIDTH and ON_SLAVE_DIR attributes define the constraints for the signal when present in a slave bus interface.

- When you load any IP-XACT XML file in GenSys, following four extension tables are created in GenSys GUI:
  - component (in Component View)
  - design (in Design View)
  - designConfiguration (in Design View)
  - busDefinition (in Interface View)

  These extension tables may be used for debugging purposes.

- The directConnection field in the Info tab indicates that a master interface may be directly connected to a slave interface (under certain conditions).

- The attributes table in the Interface View now has two default attributes namely, maxMasters and maxSlaves. The maxMasters attribute indicates the maximum number of masters that the bus supports. The maxSlaves indicates the maximum number of slaves that the bus supports. Refer to the figure given below that shows these attributes:



- The busDefinition table in the Interface View now has a new extends table. This table specifies that the bus definition is compatible with the specified bus type. Extending another bus type may result in change of signal definition and addition of new signals in the existing bus definition. However, the signals in the original bus may not get deleted, but will be marked illegal to prevent their use.

  IPXACT recognizes two independent entities, bus definition and abstraction definition. In GenSys, you cannot create a bus definition independently. Therefore, when you create an interface, both bus definition and abstract definition are created together.

- In the Ports table of a component, if the default value is incomplete, GenSys completes it with the last bit (MSB bit). For example, for a port of 10 bits, if the default value is 11001, it is extended as 1111111001. In the Ports table, the value is shown as 11001 which is elaborated to 1111111001 during RTL Generation.

Tasks in the IP Packaging Flow

In this flow, you can perform the following tasks:

- Setting the Library Path
- Loading the RTL of an IP
- Updating Interface Definitions
- Specifying Packaging Options
- Generating the Packaged View
- Checking the RTL Against the Packaged View
- Performing Incremental Packaging
- Performing Hierarchical Packaging
- Viewing Reports

Setting the Library Path

For details, see Setting the Library Path.

Loading the RTL of an IP

To create an IP-XACT view of an IP, you must first load the RTL of that IP in GenSys. This process is known as RTL import.

For information on importing RTL, see the following topics:

- Importing RTL Through GUI
- Importing RTL by Using Tcl Command

Specifying Design or Component Level Packaging

To create a packaged view of a component, set the -components argument of the load_rtl Tcl command to true.

Do not specify this argument if you want to create a packaged view of a design.

Updating Interface Definitions

Once you load the RTL of an IP, you can update its interface definitions by:

- Using the GUI Menu to Create Interfaces
- Using the Canvas View to Create Interfaces On-the-Fly
- Using the Tabular View to Create Interfaces
- Using Tcl to Create Interfaces
- Using MS Excel to Create Interfaces
- Using the Interfaces Suggested in GenSys
- Mapping Interfaces

Specifying Packaging Options

While creating the IP-XACT view of a design or component, you can specify packaging options, such as specifying file set information, preserving the RTL language of design or component, and interpreting pragma information from the RTL.

Specify these options by using the set_packaging_options Tcl command. You can also use the set_rtlimport_option Tcl command to specify certain packaging options.

For details on this command, refer to *GenSys Tcl Commands Reference Guide*.

Generating the Packaged View

After Updating Interface Definitions and Specifying Packaging Options, generate the packaged view of the IP in any of the following ways:

- Click the *File -> Export* or *File -> Export All* menu option.

  The *Choose Destination Directory* dialog appears in which you specify the directory and the package format, such as IP-XACT 1.2, IP-XACT 1.4, or IP-XACT ieee_2009.

- Use the export_all or the export_data Tcl commands.

  For details on these commands, refer to the *GenSys Tcl Commands Reference Guide*.

Checking the RTL Against the Packaged View

It may happen that after generating the packaged view of an IP, you may modify the RTL of the IP. In such cases, the RTL and the packaged view of the IP get out-of-sync.

In such cases, you can check the type of changes you made in the RTL and decide if you need to regenerate the packaged view.
To check the RTL changes, use either the check_rtl Tcl command or the Check RTL menu option from the GUI. For details on the check_rtl Tcl command, refer to *GenSys Tcl Commands Reference Guide*.

GenSys stores all the RTL changes in the check_rtl.log file.

Viewing the check_rtl Log File

This file shows differences between the RTL view and packaged view available in the library.

The differences are in the form ports and parameters changes in the following format:

```
=====================================Differences for Component
withname:c1,vendor:v,library:l, version:1.0Ports :Added ports : Port6,
Port7Modified_ports : Port1, Port2Deleted_ports : Port3, Port5
Parameters :Added_parameters : param6 param7Modified_parameters : param1
param2Deleted_parameters : param3
param5=====================================
```

Performing Incremental Packaging

Incremental packaging means to incrementally generate the packaged view whenever you modify the RTL.

For example, after Checking the RTL Against the Packaged View, if you see some differences in the RTL with respect to the packaged view, you can regenerate the packaged view to make it in sync with RTL. For details on generating the packaged view, see Generating the Packaged View.
To update the packaged view with respect to the RTL changes, set the -update_component argument of the load_rtl Tcl command to TRUE to update packaged IP with respect to its RTL view

When you set this argument to TRUE, the following occurs:

- GenSys searches for the currently loaded component, design, or subsystem that has the same VNLV as specified by the -version, -vendor, -name, and -library arguments of the load_rtl Tcl command.
- If such an object is found, GenSys updates that object by merging information from the RTL view.

Performing Hierarchical Packaging

Hierarchical packaging means creating a packaged view of a design.

The term *hierarchical* is used in this context because a design contains many hierarchies within it. Therefore, when you package a design, GenSys packages the design along with all its hierarchies.

For details on packaging, see Generating the Packaged View.

Viewing Reports

Once the packaged view of an IP is ready, you can view the following reports:

- Design Summary

  This report is useful if you have created the packaged view of a design.

- Interface Summary

  This report provides a summary and details of interfaces instantiated on the IP for which you created the packaged view.

The RTL Restructuring Flow

Use this flow to modify the design hierarchy. You can perform RTL-based assembly in this flow.

However, you cannot make any other design modifications other than hierarchy manipulations in this flow. To make other design modifications, use The Structured Assembly Flow.

The Need to Modify the Design Hierarchy

Hierarchy modification is required if the current design hierarchy is resulting in issues, such as:

- Timing is not met properly.

- Power performance is getting impacted.
- Congestion

Tasks in the RTL Restructuring Flow

In this flow, you can perform the following tasks:

- Loading the RTL of an IP
- Moving Instances
- Adding a Hierarchy
- Dissolving a Hierarchy
- Grouping
- Removing Floating Ports After Hierarchy Manipulation
- Reverting Hierarchy Manipulation Operations
- Reverting Hierarchy Manipulation Operations
- Optimizing Ports Generated on Subsystem Boundaries
- Viewing Library Cells in a Hierarchy
- Comparing Original and New Hierarchies
- Cloning Instances
- Adding Clones in the Hierarchy of Instances of the Specified Design Masters
- Solidifying Glue Instances and Generate Blocks
- Adding Planes
- Performing Equivalence Checking
- Checking the Design Health
- Debugging and Analyzing the Design
- Generating RTL
- Viewing Reports

Loading the RTL of an IP

To modify the hierarchy of an IP, you must first load the RTL of that IP in GenSys. This process is known as RTL import.

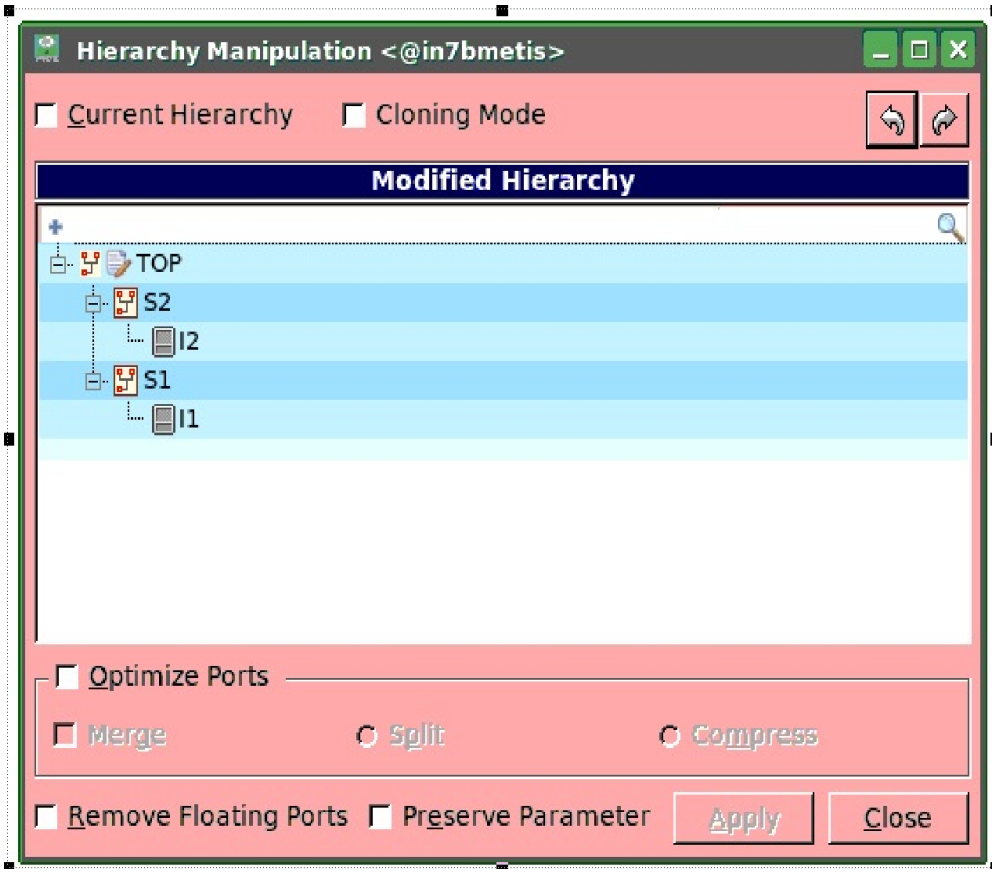
For information on importing RTL, see the following topics:

- Importing RTL Through GUI
- Importing RTL by Using Tcl Command

Moving Instances

You can move instances from one hierarchy to another hierarchy.

While moving instances, GenSys preserves the entire connectivity. It may also create some ports on sub systems to preserve connectivity.
Consider the hierarchy shown in the following figure:



In the above dialog, to move the I1 instance from the S1 sub system to the S2 subsystem, perform the following steps:

1. Select the I1 instance, and press <*Ctrl* + *X*> keyboard keys.

the I1 instance appears in bold.

2. Select the S2 hierarchy where do you want to move the I1 instance.

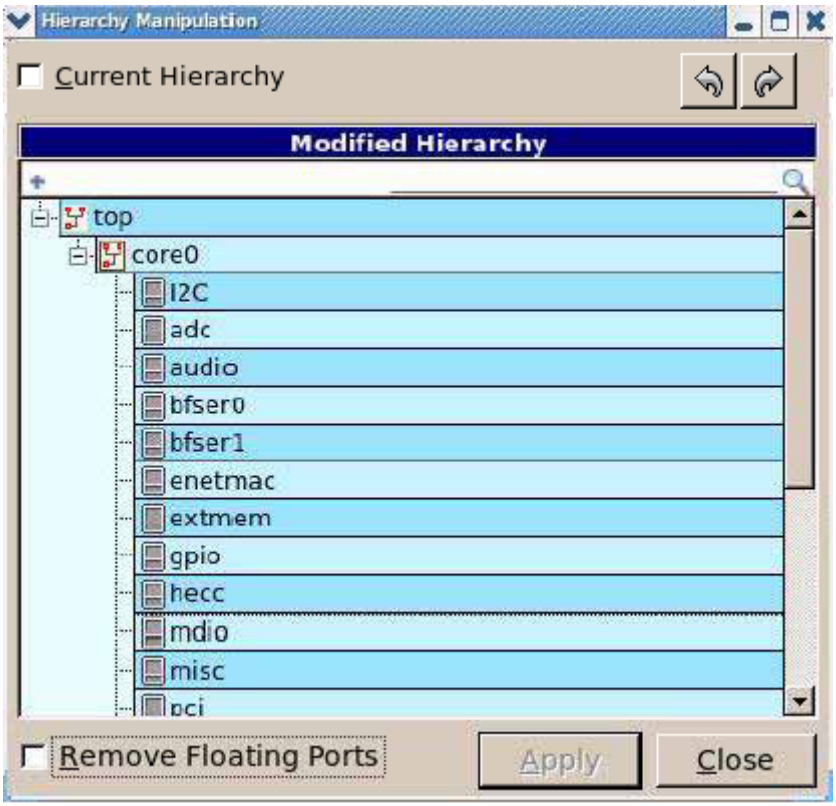
1. Select <*Ctrl*+*V*> keyboard key to move I1 in the S2 hierarchy.
2. (Optional) To delete hanging ports created in the above step, select the Remove Floating Ports check box in the above dialog.
3. (Optional) To preserve parameterized expressions of the pins of the instance being moved, select the Preserve Parameters check box. Only the expressions containing parameters that are controlled from the diverge point are preserved. To preserve parameters in the destination side, new parameters are added to the destination side..

   For more information on this operation, refer to the move_instance Tcl command in the GenSys Tcl Commands Reference Guide.
4. Click the *Apply* button.

After performing these steps, the I1 instance is moved from the S1 to the S2 hierarchy.

Adding a Hierarchy

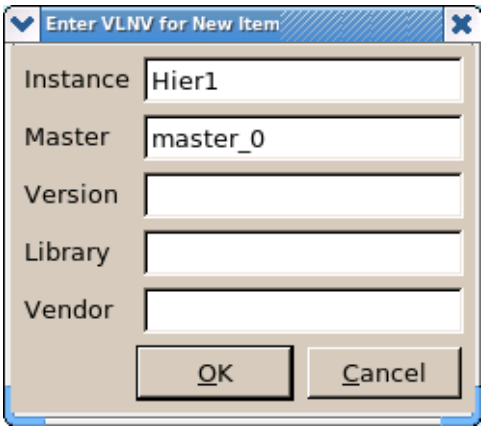
Consider the design hierarchy as shown in the following figure:



In the above dialog, add a new hierarchy by performing the following steps:

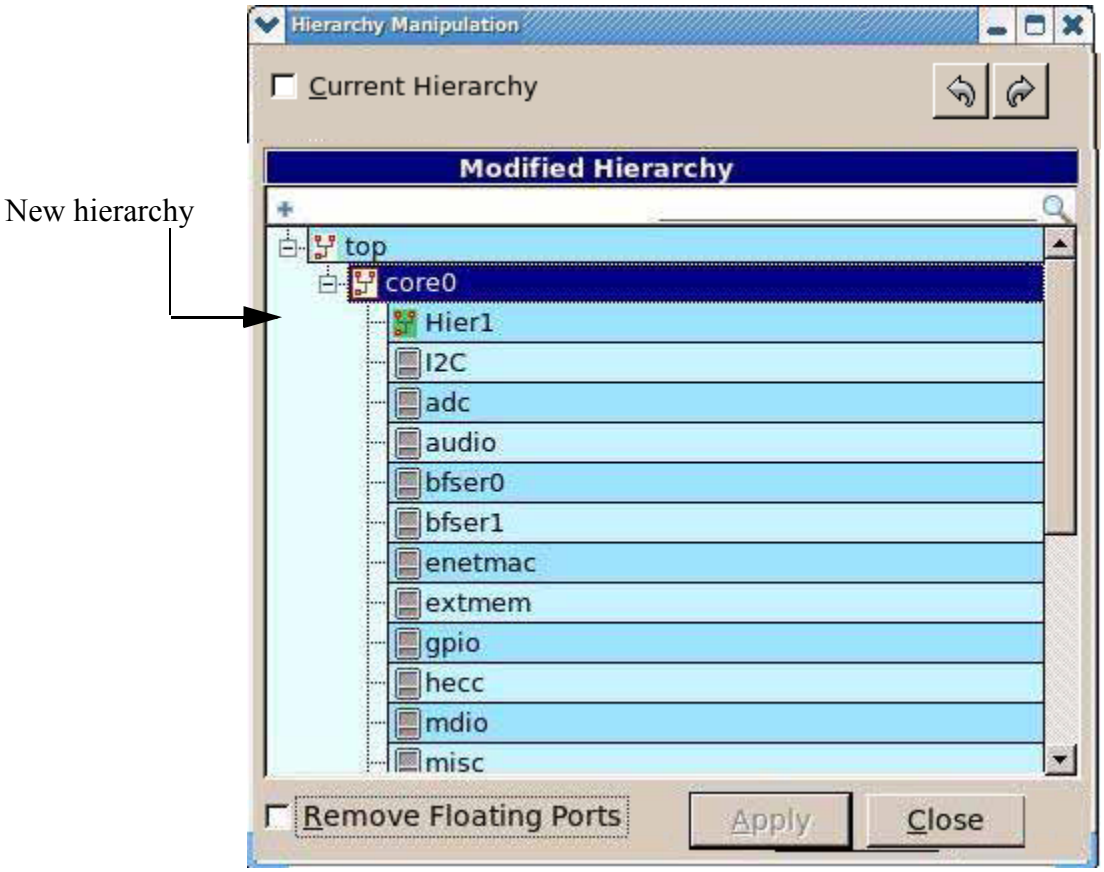
1. Select the subsystem (say *core0*) under which you want to add a new hierarchy.
2. Right-click on the selected subsystem and select the *Insert Hier* option from the shortcut menu.

   The *Enter VLNV for New Item* dialog appears, as shown in the following figure:



3. In the above dialog, specify the details, such as instance name, master name, and VLNV of the subsystem representing the new hierarchy.
4. Click the *OK* button to close the above dialog.

   The new hierarchy appears in the *Hierarchy Manipulation* dialog, as shown in the following figure



5. Compare the hierarchy changes with the original design hierarchy by selecting the *Display Current Hierarchy* checkbox.

   For details, see Comparing Original and New Hierarchies.

6. To save the hierarchy changes, click the *Apply* button.

   Note:
   You can move instances across hierarchies by pressing the Ctrl+X (cut) and Ctrl+V (copy) keys.

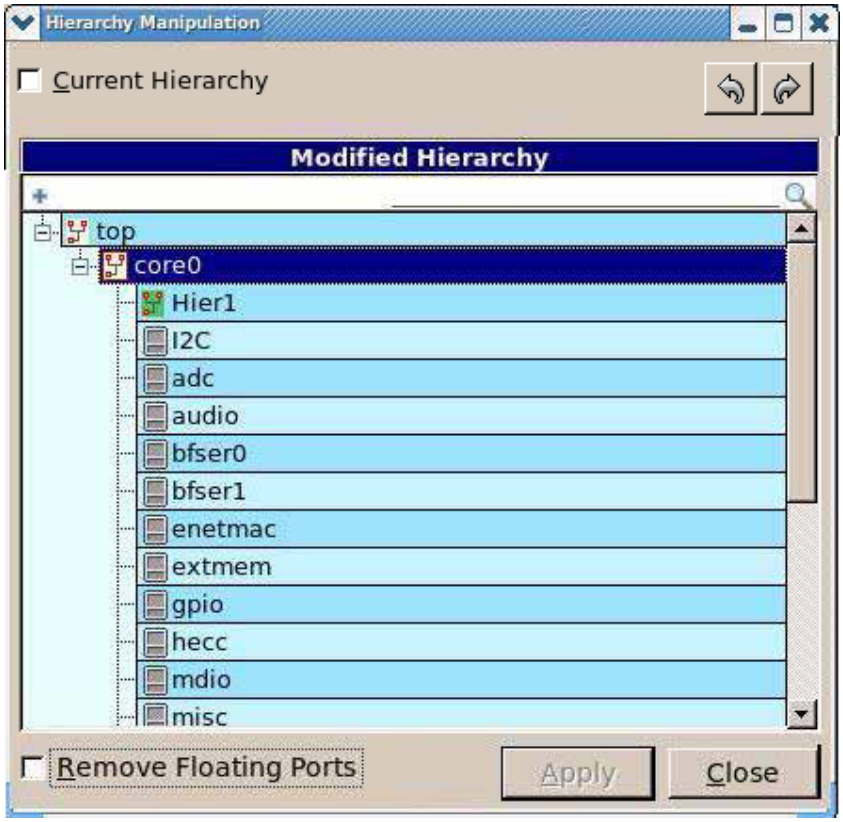
Dissolving a Hierarchy

To dissolve or remove a hierarchy, right-click on the hierarchy in the *Hierarchy Manipulation* dialog and select the *Dissolve Hier* option from the shortcut menu.

This option is disabled for items that have RTL glue components as their children and also if their master is open in the modular schematic view.

Grouping

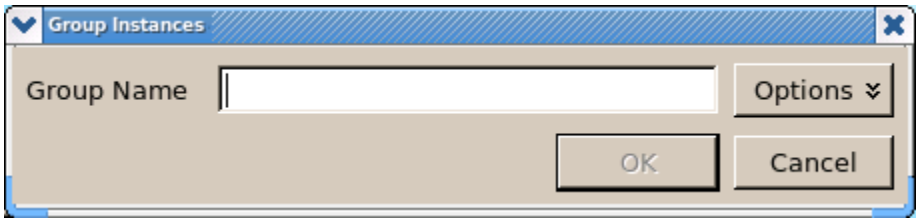
You can create and assign groups to the subsystems of the currently loaded design. For example, consider the dialog shown in the following figure:



In the above dialog, to assign a group (say grp) to Hier1 and audio, perform the following steps:

1. Select Hier1 and audio by keeping the *Ctrl* keyboard key pressed.
2. Right-click on the selection and select the *Group* option from the shortcut menu.

   The *Group Instances* dialog appears, as shown in the following figure:

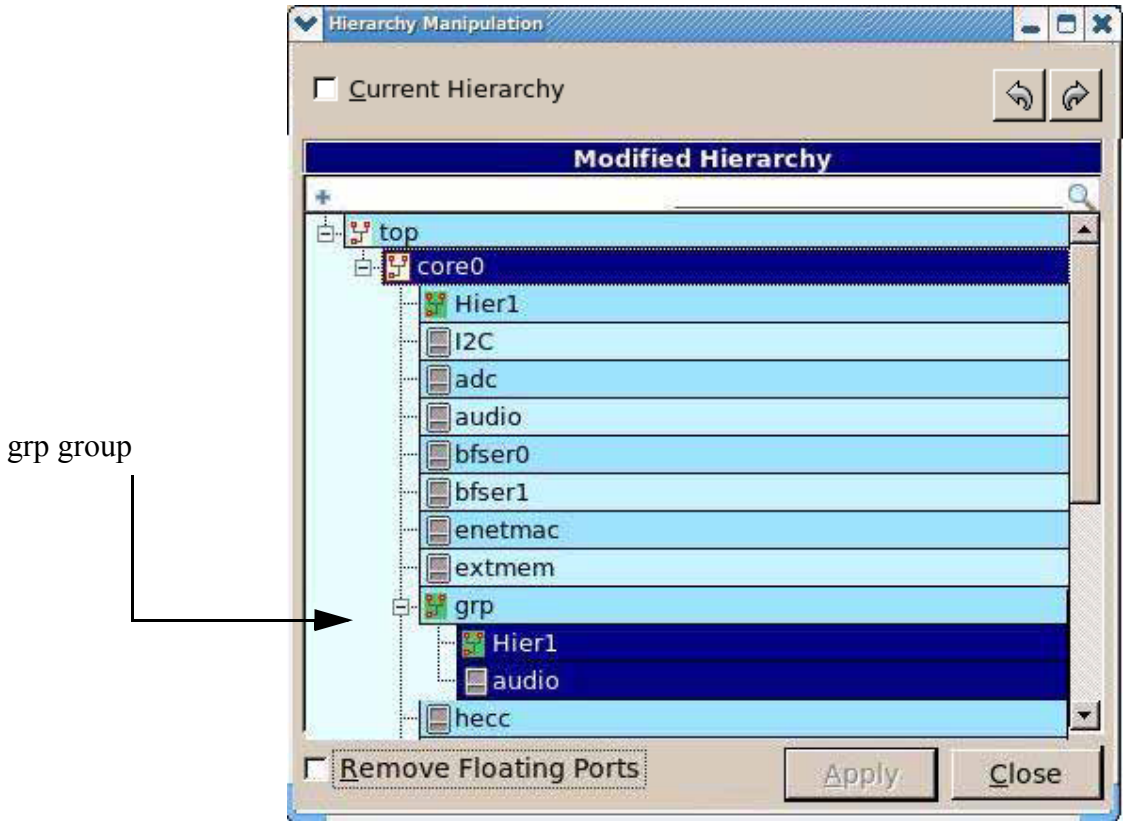


3. In the above dialog, specify the group name (in this case grp).

   To provide grouping options, such as group master name and port naming format, click the *Options* button. This displays the *Group Instances* dialog in which you can specify the grouping options.

4. Click the *OK* button.

After performing the above steps, Hier1 and audio appear under the grp group hierarchy, as shown in the following figure:



Removing Floating Ports After Hierarchy Manipulation

After hierarchy manipulation, such as Dissolving a Hierarchy and Grouping, there may be floating ports in the hierarchy, that is ports with no connection.

To delete such ports from the modified design hierarchy, select the *Remove Floating Ports* option in the *Hierarchy Manipulation* dialog.

Reverting Hierarchy Manipulation Operations

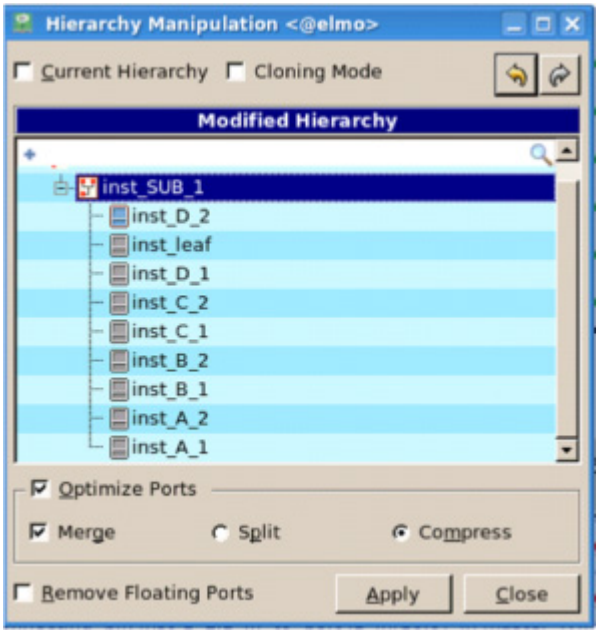
During After hierarchy manipulation, such as Adding a Hierarchy, Dissolving a Hierarchy and Grouping, you can undo and redo changes by clicking the undo and redo buttons in the *Hierarchy Manipulation* dialog.

After undo and redo, click the *Apply* button to save the changes. Once you click this button, you cannot revert the undo and redo operations.

Optimizing Ports Generated on Subsystem Boundaries

After the hierarchy is manipulated, by selecting the Optimize Ports check box, GenSys optimizes the ports generated on subsystem boundaries as part of various hierarchy manipulation and connection creation operations.

There are three types of port optimization: Merge, Compress, and Split.



For more on information on port optimization, refer to the optimize_ports Tcl command documentation in the GenSys Tcl Commands Reference Guide.

Viewing Library Cells in a Hierarchy

If any hierarchy contains library cells, you can view them by right-clicking on that hierarchy in the above dialog and selecting the Expand Lib Cells option from the shortcut menu.

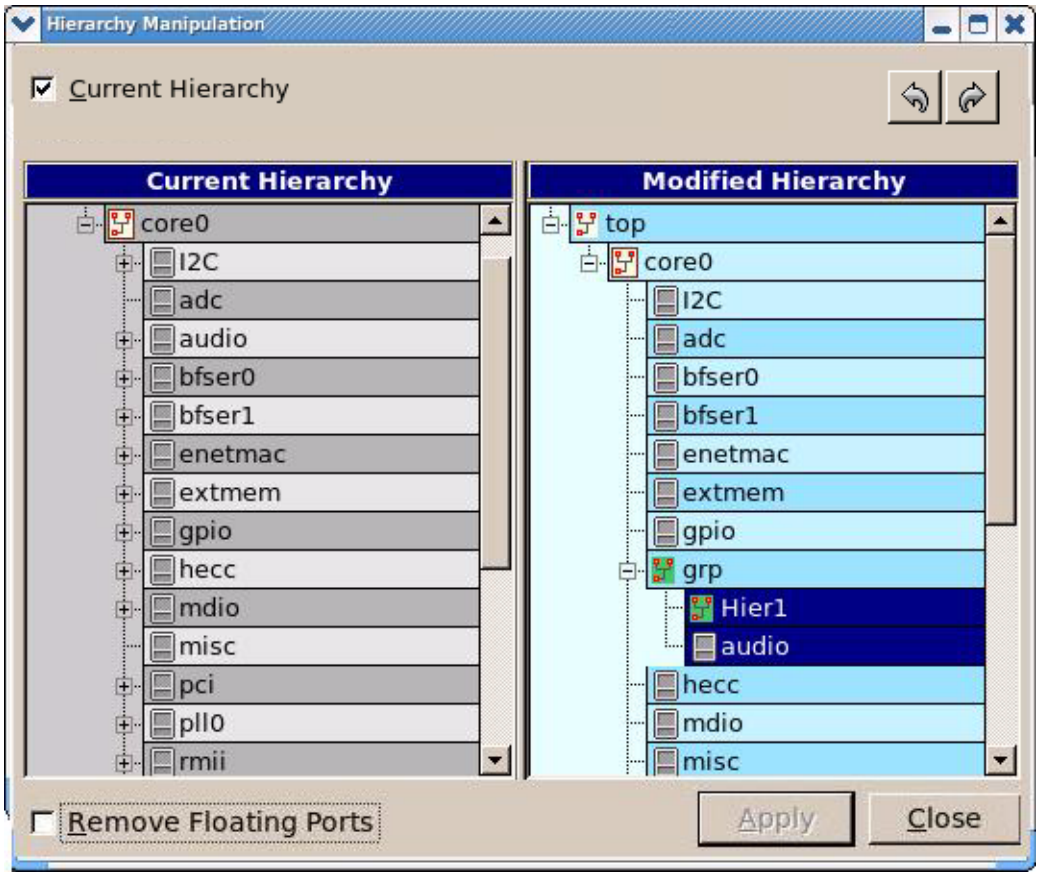
To hide library cells in a hierarchy, right-click on that hierarchy and select the Collapse Lib Cells option from the shortcut menu.

These options appear only if library cells are present in the selected hierarchy.

Comparing Original and New Hierarchies

After making hierarchy changes by Adding a Hierarchy, Dissolving a Hierarchy, and Grouping, you can compare the changes with the original design. This comparison enables you to check if the hierarchy changes are as expected.

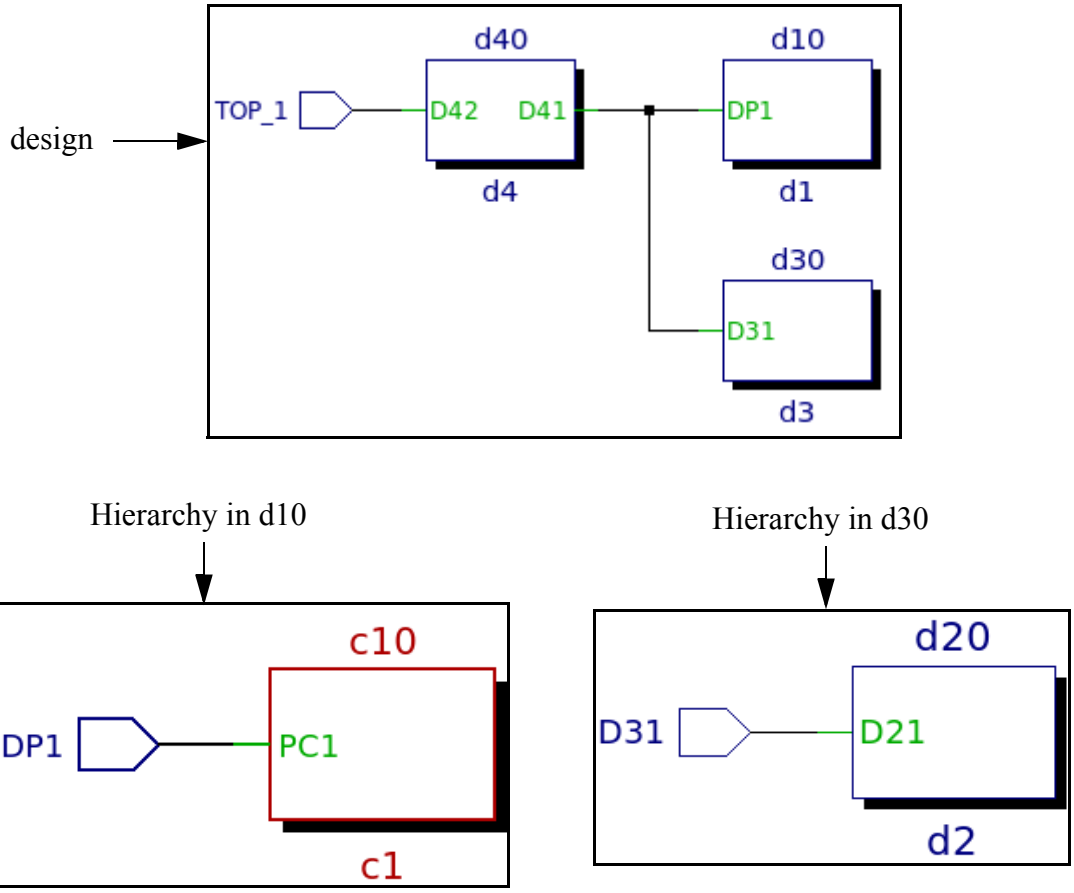
To do such comparison, select the *Current Hierarchy* checkbox in the *Hierarchy Manipulation* dialog. When you select this checkbox, GenSys shows the hierarchy comparison between the new and the original design, as shown in the following figure:



Cloning Instances

Instance cloning is required for physical design reasons to reduce congestion and long routing wires spanning across hierarchies.

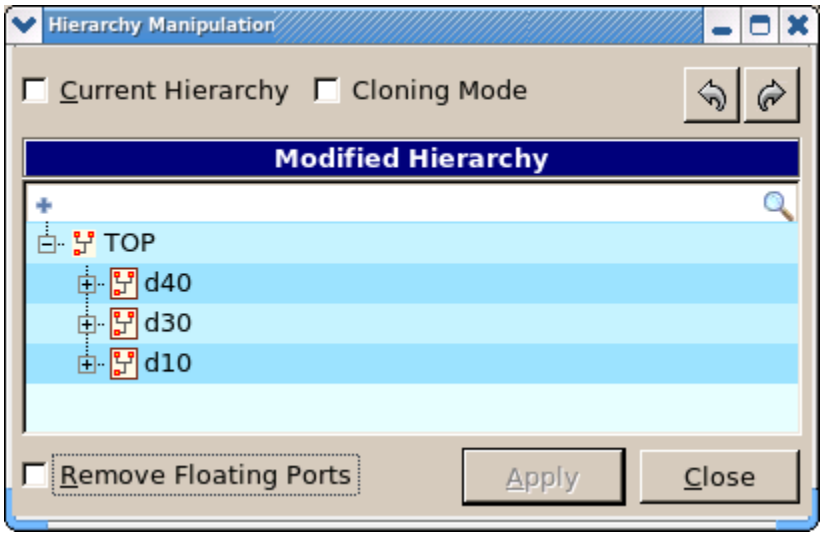
To enable instance cloning, select the Cloning Mode check box in the Hierarchy Manipulation dialog.
Consider the following design:



To add clones of d40 in d10 and d30 in the above design, perform the following steps:

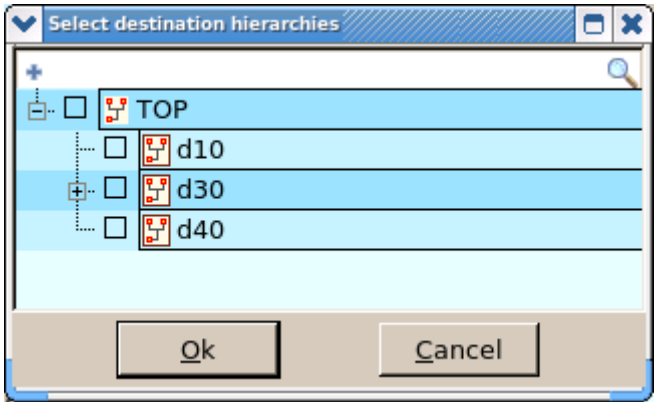
1. Select Tools -> Hierarchy Manipulation menu option.

   The following dialog appears:



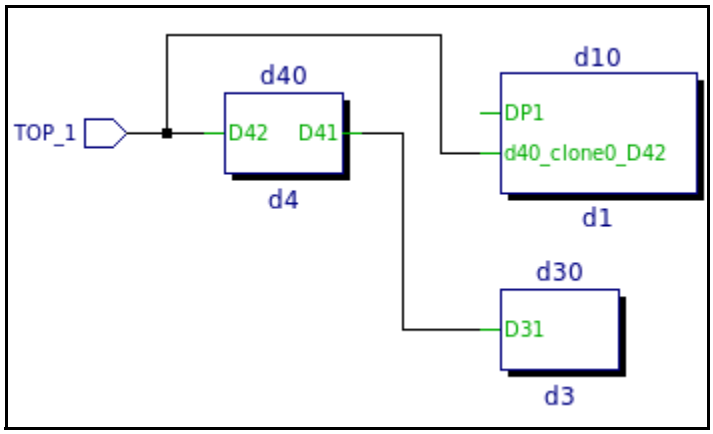
2. In the above dialog, select the Cloning Mode check box.
3. Right-click on d40 and select the Clone -> Specify Hierarchy option from the shortcut menu.

   The following dialog appears:

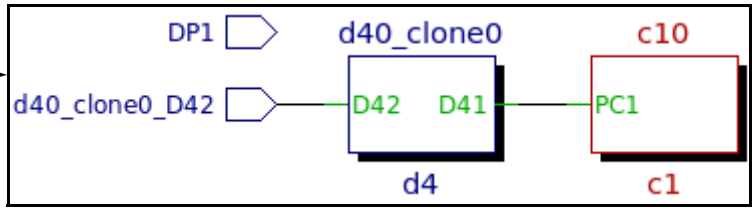


4. In the above dialog, select the d10 check box and click the OK button.
5. Click the Apply button in the Hierarchy Manipulation dialog to save the changes.

   The design changes to the following:

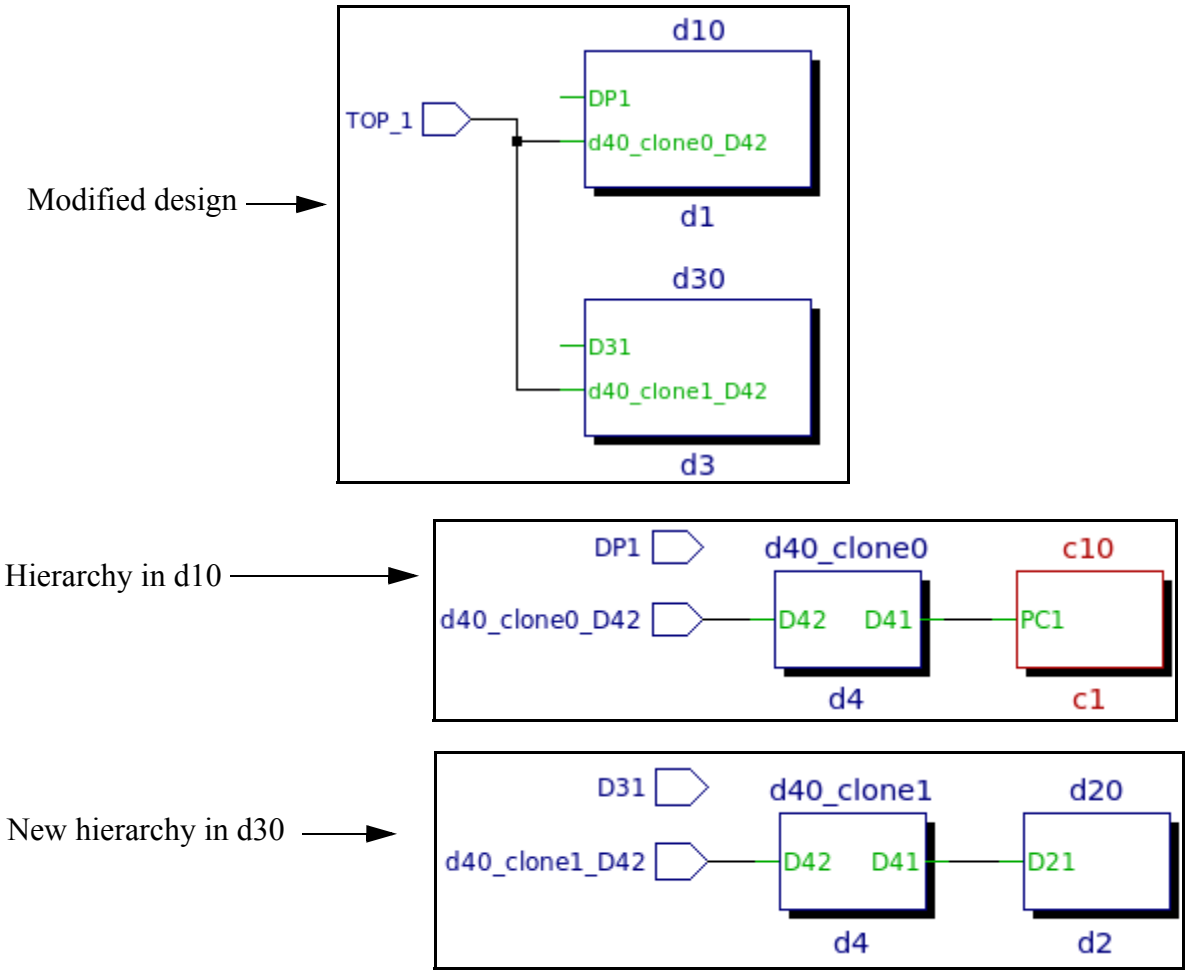


New hierarchy in d10



6. In the Hierarchy Manipulation dialog, right-click again on d40 and select the Clone -> Specify Hierarchy option from the shortcut menu.
7. Select the d30 check box and click the OK button.
8. Click the Apply button in the Hierarchy Manipulation dialog.

   The design changes to the following:



You can also perform cloning in the above steps by using the following Tcl command:

```
clone_ip -name TOP/d40 -to {TOP/d10 TOP/d30}
```

For details on the clone_ip command, refer to *GenSys Tcl Commands Reference Guide*.

Adding Clones in the Hierarchy of Instances of the Specified Design Masters

You can specify design masters whose instances should contain the specified clones.

To add clones in such cases, perform the following steps:

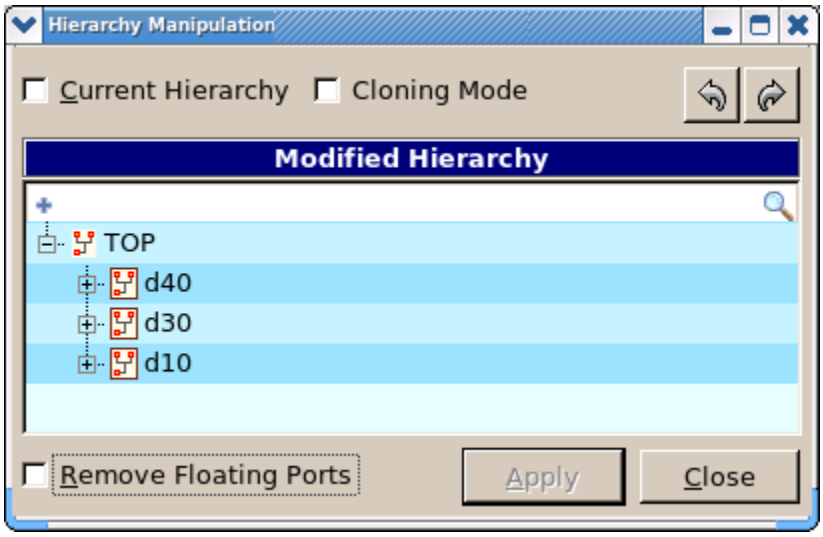
1. Use the set_physical_units Tcl command to specify the design masters whose instances should contain clones.

Suppose you specify the following command to specify the d1 and d3 masters so that their instances, d10 and d30, respectively, should contain clones:

```
set_physical_units -names {d1 d3} -append
```

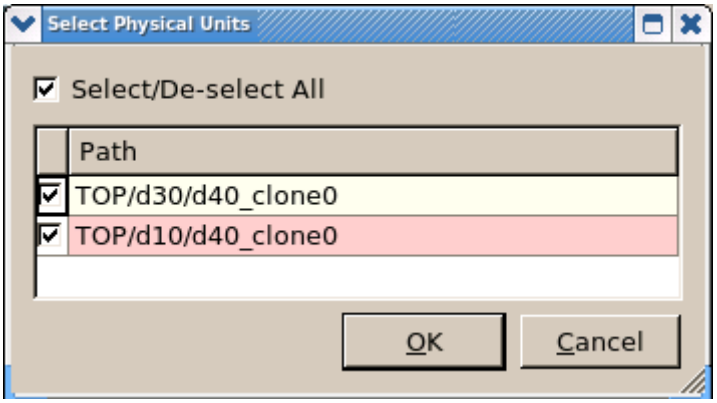
2. Select the Tools -> Hierarchy Manipulation menu option.

   The following dialog appears:



3. In the above dialog, select the Cloning Mode check box.
4. To create the clone of an instance (say d40), right-click on that instance and select the Clone -> To Physical Units option from the shortcut menu.

   The following dialog appears:



The above dialog shows the clone names of the instance selected in this step.

These clone names appear under the hierarchy of instances whose masters are specified by the set_physical_units Tcl command in step 1.

5. In the above dialog, select clone you want to generate.
6. Click the OK button.

After performing the above steps, the clones selected in step 1 are generated.

Adding Planes

For details, see Working with Planes.

Performing Equivalence Checking

An SOC contains sub-systems that further contains many instances and complex connectivity among them. Therefore, after restructuring, user should be able to verify that the design before (reference design) and the design after performing restructuring (implementation design) are functionally equivalent.

The write_formality_script GenSys Tcl command generates a formality script that can be used to validate restructured RTL using formality.

Checking the Design Health

You can check the health of a design in terms of its connection details, such as:

- The instance name, pin name, interface name, and pin direction in the source and destination of the connection.
- Whether the terminals of the instances in the design are connected, unconnected, or tied-off to a particular value.
- The connection status
- The Tcl command used to create the connection.

This feature generates a report that displays connectivity after resolving overlapping connections. It shows whether terminals of these instances are connected, unconnected, or tied-off to a particular value.

To generate hierarchical or flat RTL Health Check Report for the current design or any sub-system design inside it. The RTL Health Check report can be generated as:

- Hierarchical RTL Health Check: To be able to generate RTL Health Check report at each level in the hierarchies so that you can check the connectivity summary at each level in the design hierarchy.

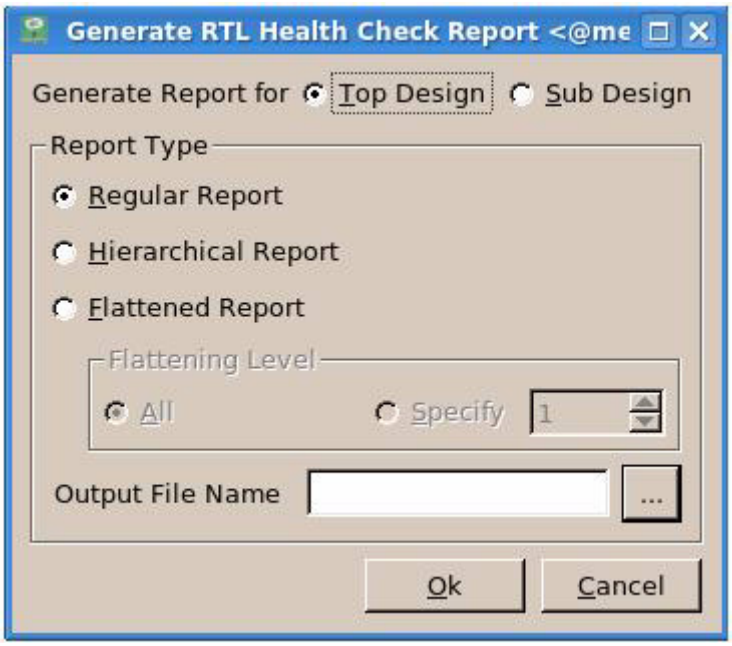
- Flat RTL Health Check: To be able to generate flat RTL Health Check report for top-level SoC, or for any sub-system inside it. This enables you to check final connectivity summary at flat level.

To generate the *RTL Health Check* report, perform either of the following and also read the Key Points for the RTLHC Report:

- Using the Report -> RTLHC Report Menu Option
- Using the create_report Tcl Command

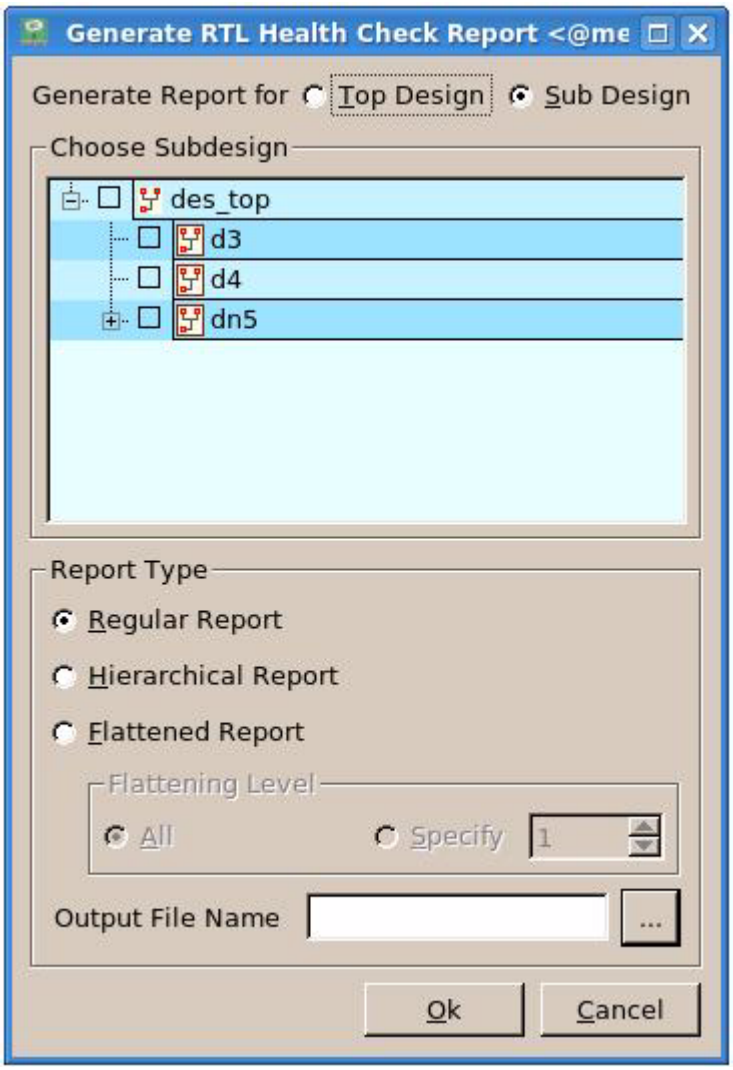
Using the Report -> RTLHC Report Menu Option

You can generate the RTLHC report from the GenSys GUI by clicking the *Report -> RTLHC Report* menu option. The Generate RTL Health Check Report dialog box appears.



By default, the RTL Health Check report is generated for the Top Design. The Regular Report does not have hierarchy information.
For the Flattened Report, options available either to flatten all levels, or up to a certain level specified by the user. The output file name is optional, and if provided, RTL health check report file (csv) is generated with the specified name. If there are multiple files being generated, as is the case for the Hierarchical Report, the design name is appended to the specified file name. GenSys generates the design health report at the following location:
$CWD/report/rtl_health_check_<design_name>.csv

If you need to generate the RTL Health Check report for a sub-design, without loading the design as a top-level design in GenSys, select the Sub Design radio button. This opens a design tree from which you can select the sub-design.



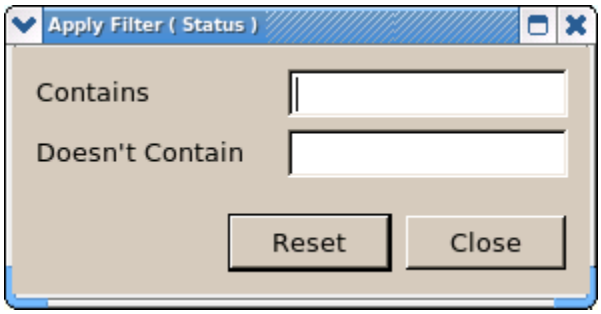
Select the sub-design, the type of report, and then click the Ok button. The RTL Health Check table appears with the relevant information, as shown in the following figure:

	Instance	Interface	Pin	PinDirection	Destinatio	Destinatior	DestinationPin	Status
207	FABRIC0	APB_S3	PSEL6_S3	in				not connected
208	FABRIC0	APB_S3	PSEL7_S3	in				not connected
209	FABRIC0	APB_S3	PWDATA_S3[31:0]	in				not connected
210	FABRIC0	APB_S3	PWRITE_S3	in				not connected
211	FABRIC0	AXI_S0	ARADDR_S0[31:0]	out	CC_CFG00	AXI	ARADDR_c[63:32]	connected
212	FABRIC0	AXI_S0	ARBURST_S0[1:0]	out	CC_CFG00	AXI	ARBURST_c[1:0]	connected
213	FABRIC0	AXI_S0	ARCACHE_S0[3:0]	out	CC_CFG00	AXI	ARCACHE_c[3:0]	connected
214	FABRIC0	AXI_S0	ARID_S0[3:0]	out	CC_CFG00	AXI	ARID_c[3:0]	connected
215	FABRIC0	AXI_S0	ARLEN_S0[3:0]	out	CC_CFG00	AXI	ARLEN_c[3:0]	connected
216	FABRIC0	AXI_S0	ARLOCK_S0[1:0]	out	CC_CFG00	AXI	ARLOCK_c[1:0]	connected
217	FABRIC0	AXI_S0	ARPROT_S0[2:0]	out	CC_CFG00	AXI	ARPROT_c[2:0]	connected
218	FABRIC0	AXI_S0	ARREADY_S0	in	CC_CFG00	AXI	ARREADY_c	connected
219	FABRIC0	AXI_S0	ARSIZE_S0[2:0]	out	CC_CFG00	AXI	ARSIZE_c[2:0]	connected
220	FABRIC0	AXI_S0	ARVALID_S0	out	CC_CFG00	AXI	ARVALID_c	connected
221	FABRIC0	AXI_S0	AWADDR_S0[31:0]	out	CC_CFG00	AXI	AWADDR_c[63:32]	connected
222	FABRIC0	AXI_S0	AWBURST_S0[1:0]	out	CC_CFG00	AXI	AWBURST_c[1:0]	connected
223	FABRIC0	AXI_S0	AWCACHE_S0[3:0]	out	CC_CFG00	AXI	AWCACHE_c[3:0]	connected

In the above table, you can filter the status column to check the pins that are connected or not connected.
For example, to view details of unconnected ports, perform the following steps:

1. Right-click on the header of the *Status* column and select the *Filter Options* option from the shortcut menu.

   The following dialog appears:



2. In the above dialog, type *not connected* in the *Contains* text box.
3. Click the *Close* button.

After performing the above steps, the details of only unconnected ports appear in the *RTL Health Check* report.

Using the create_report Tcl Command

Use the following Tcl command to generate the RTLHC report:

```
create_report -category {debug} [-hierarchy {TRUE|FALSE}] [-design { name
of the design }] [-flatten {TRUE | FALSE|<integer value>}] [-file <name
of the report file>]
```

For details of this Tcl command, refer to the *GenSys Tcl Commands Reference Guide*.

Key Points for the RTLHC Report

Note the following points:

- GenSys automatically elaborates the design during RTL Health Check. This helps in creating adhoc connections and evaluating parameterized ports before RTL health check.
- In the Rtl_Health_Check table, multiple fan-out are displayed in separate rows. For example, the following output-to-input connections are displayed on separate rows:
  - out_sig -> i_1.in_sig
  - out_sig -> i_2.in_sig
  - out_sig -> i_3.in_sig

By default, the ChangeName and ChangeDate columns display the oldest datasource name and date, respectively. You can use the Tcl command, set_view_options, to display the latest datasource name and date. For more details, refer to the *Tcl Commands Reference Guide*.

Debugging and Analyzing the Design

After performing hierarchical changes in the design, you can debug and analyze the design in the following ways:

- Viewing Multiple Schematics
- Searching Objects in a Congested Schematic
- Traversing Through Unconnected Ports

Viewing Multiple Schematics

In the canvas view, GenSys displays a separate tab corresponding to each object appearing in the design browser.

Click on a tab to view the schematic of the corresponding object.

Searching Objects in a Congested Schematic

For details, see Searching Objects in the Schematic.

Traversing Through Unconnected Ports

For details, see Viewing Only Unconnected Ports and Pins and Navigating between Unconnected Ports and Pins.

Generating RTL

For details, see Generating RTL.

Viewing Reports

Once you have modified the design hierarchy, you can view the following reports:

- Design Summary
- Connection Summary
- Design statistics report in the Statistics Window

The Structured Assembly Flow

Use this flow to make modifications to the design loaded in GenSys.

You can modify derivative designs or build new designs from scratch in this flow.

Tasks in the Structured Assembly Flow

In this flow, you can create new designs or modify existing designs.

For details, see Building SoCs.

3

Creating Libraries

A library is a repository of objects, such as Designs, Components (IPs), and Interfaces. A designer assembles these objects to build SoCs. You can either create library objects from scratch or import them in GenSys.

For details on importing objects, see Performing RTL Import.

This chapter describes the prerequisites for library creation. It also covers details on creating library objects.

Prerequisites for Library Creation

Before creating a library, perform the following tasks:

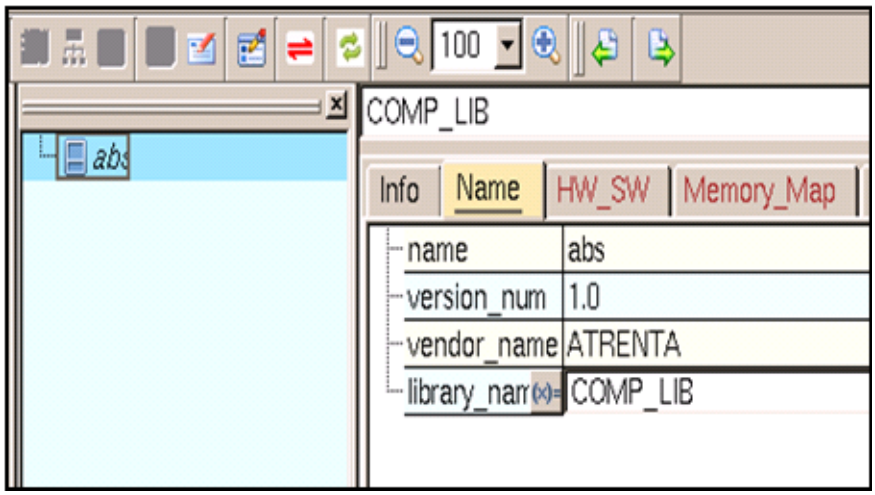
- Setting VLNV
- Setting a Library Path
- Specifying a Default Save Format

Setting VLNV

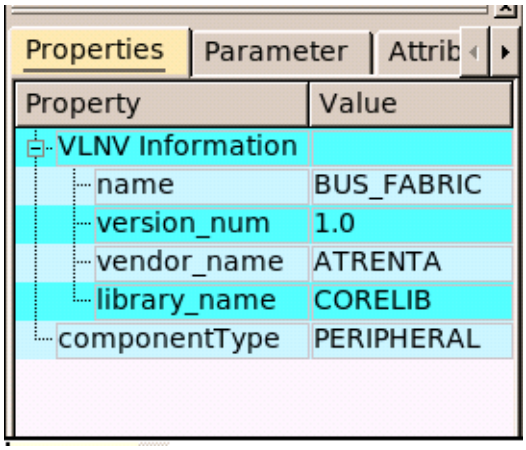
Specify a default VLNV for the library objects for which the VLNV information is not specified.

You can set a default VLNV in any of the following ways:

- By using the set_default_vlv Tcl command.
- By specifying the VLNV information while loading the RTL by using the load_rtl Tcl command.
- By specifying the VLNV information while creating a new object by using the new Tcl command.
- By specifying the VLNV information under the *Name* tab in the tabular view, as shown in the following figure:



- By specifying the VLNV information under the *Properties* tab in the *Property Editor* pane of the canvas view, as shown in the following figure:



Setting a Library Path

Set a library path to automatically load the library objects from the specified path.

Use the set_library_path Tcl command to set a library path.

Specifying a Default Save Format

Set a default format in the which the library objects should be saved.

Use the set_default_save_format Tcl command to specify a default save format.

GenSys supports the following save formats:

- IP-XACT: Use this format if the design objects are to be shared among different users. You can specify the IP-XACT version 1.2, 1.4, or IEEE 1685.
- MRS (Default): Use this format if you want to store extra information, such as glue logic.
- Tcl: Use this format if you want to create scripts to recreate designs every time you load the design in GenSys.

Building Interfaces and Interface Connections

You can create Interfaces in different ways and then map logical ports of the interface with the actual ports of Components.

Creating Interfaces

You can create interfaces in any of the following ways:

- Using the GUI Menu to Create Interfaces
- Using the Canvas View to Create Interfaces
- Using the Canvas View to Create Interfaces On-the-Fly
- Using the Tabular View to Create Interfaces
- Using Tcl to Create Interfaces
- Using MS Excel to Create Interfaces
- Using the Interfaces Suggested in GenSys

Using the GUI Menu to Create Interfaces

To create interfaces in GenSys GUI, perform the following steps:

1. Select the File-> New menu option. The New dialog appears.
2. Select the Interface option in the New dialog.
3. Click the OK button. The New File dialog appears.
4. Specify the name of the interface in the Input Name for New File: text field.
5. Click the OK button.

Using the Canvas View to Create Interfaces

For details, see Creating New Objects.

Using the Canvas View to Create Interfaces On-the-Fly

In the canvas view, you can select some actual ports of an instance and create an interface by making these ports a part of that interface. This process is known as creating interfaces on-the-fly.

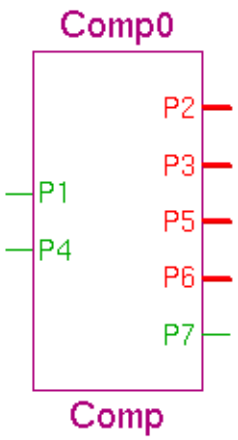
To create interfaces on-the-fly, perform the following steps:

1. Enter in the edit mode by clicking the

2. button in the schematic toolbar.

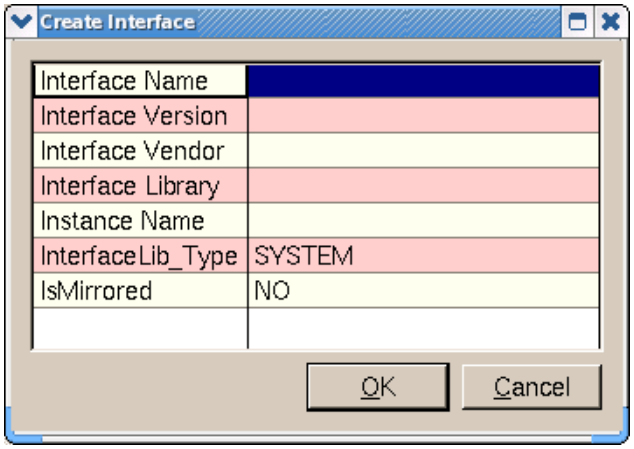
3. Select ports of a component instance or design by keeping the <*Shift*> keyboard key pressed.

   The following figure shows the P2, P3, P5, and P6 ports selected on a component instance:

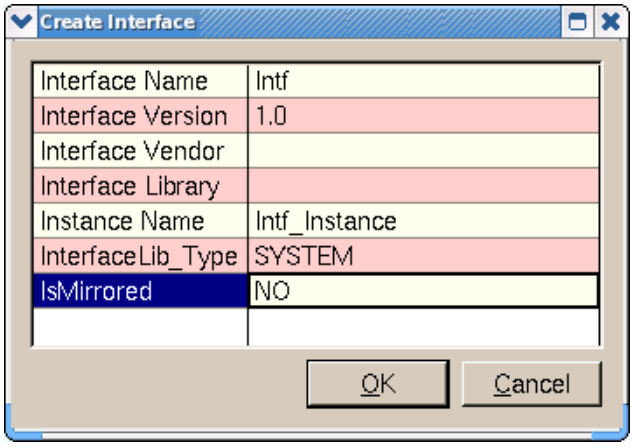


4. Right-click on the selected ports, and select the Create Interface option from the shortcut menu.

   The Create Interface dialog appears, as shown in the following figure:

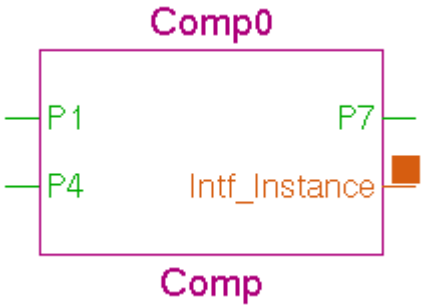


5. In this dialog, specify the mandatory details, such as interface name and instance name, and the optional details, as shown in the following figure:



6. Click the OK button.

After performing the above steps, the selected ports become a part of the interface, and the instance of that interface appears instead of these ports. This is shown in the following figure:



Using the Tabular View to Create Interfaces

To create an interface in the tabular view, perform the following steps:

1. Click the

2. button in the GUI to enter into the tabular view.
3. Perform the steps mentioned in Using the GUI Menu to Create Interfaces.

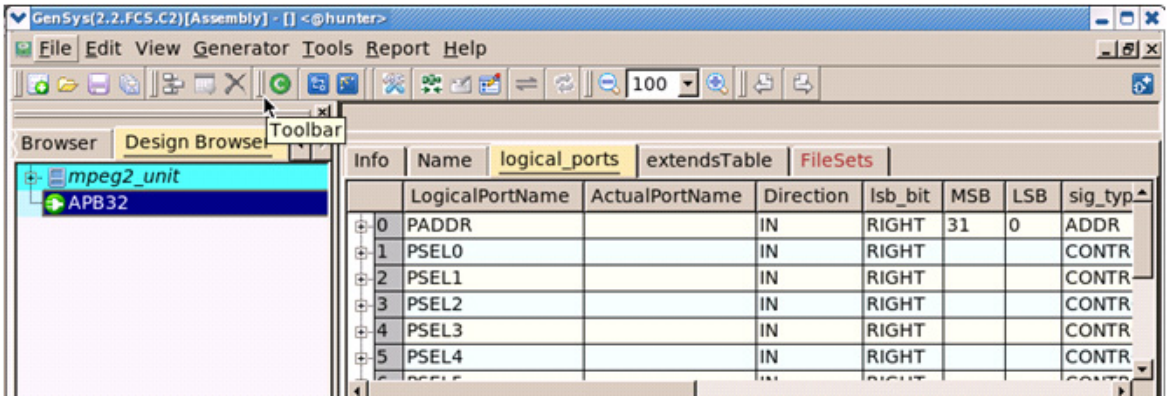
   Consider that you specify the name of the interface as APB32.

After performing the above steps, the interface appears under the Design Browser tab.

4. Click the Name tab.
5. Specify the interface properties under this tab.
6. Click the Name tab.
7. Specify the VLNV information under this tab.

   For details on VLNV, see Using VLNV to Identify an Object.
8. Click the logical_ports tab.
9. Right-click in the area under this tab, and select the Insert Rows option from the shortcut menu.

   A new row appears under this tab.
10. Specify the logical ports details, as shown in the following figure:



11. In the above table, specify the actual port name, MSB, and LSB in case of vector ports. For scalar ports, leave the corresponding columns blank.
12. Specify if the port is optional or required in the *Optional* column of the above table.

    When a port is marked as required, GenSys reports an error if you do not specify the actual port name for that port when the interface is used on a IP.
13. Select the *File -> Save* menu option to save the interface and make it a part of the library.

Using Tcl to Create Interfaces

In Tcl, use the following commands to create and specify interface details:

- Use the new interface <interface-name> Tcl command to create an interface.

- Use the set_interface_port command to add ports to the interface.
- Use the save Tcl command to save the interface and make it a part of the library.

For details on these Tcl commands, refer to the *GenSys Tcl Commands Reference Guide*.

Using MS Excel to Create Interfaces

You can create an interface by specifying its details in an MS Excel spreadsheet (XML SpreadSheet 2003), and then importing that sheet in GenSys.

The following figure shows such spreadsheet:

		VLNV			Ports						
Vendor	Library	Name	Version	Description	Logical Name	Width	LSB_Bit	Align	Signal Type	Direction	Presence
ATRENTA	ATRENTA_LIB	interface_UART_IO	1.0.0	UART IO Interface	UART_I	112	RIGHT	LSB	DATA	IN	OPTIONAL
					UART_O	112	RIGHT	LSB	DATA	OUT	OPTIONAL
					UART_OE_N	112	RIGHT	LSB	CONTROL	OUT	OPTIONAL
ATRENTA	ATRENTA_LIB	interface_TIMER_IO	1.0.0	Timer IO Interface	TIMER_I	112	RIGHT	LSB	DATA	IN	OPTIONAL
					TIMER_O	112	RIGHT	LSB	DATA	OUT	OPTIONAL
					TIMER_OE_N	112	RIGHT	LSB	DATA	OUT	OPTIONAL
ATRENTA	ATRENTA_LIB	TCM	1.0.0	TCM Interface	TCMAdrs	18	RIGHT	LSB	ADDR	OUT	REQUIRED
					TCMWData	32	RIGHT	LSB	DATA	OUT	REQUIRED
					TCMRData	32	RIGHT	LSB	DATA	IN	REQUIRED
					PhyTCMSize	4	RIGHT	LSB	CONTROL	IN	REQUIRED
					TCMEn		RIGHT	LSB	CONTROL	OUT	REQUIRED
					TCMWEn	4	RIGHT	LSB	CONTROL	OUT	REQUIRED
ATRENTA	ATRENTA_LIB	APB32	1.0.0	Atrenta APB32 Version	PADDR	32	RIGHT	LSB	ADDR	IN	REQUIRED
					PWDATA	32	RIGHT	LSB	DATA	IN	REQUIRED
					PSEL0		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PSEL1		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PSEL2		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PSEL3		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PSEL4		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PSEL5		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PSEL6		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PSEL7		RIGHT	LSB	CONTROL	IN	OPTIONAL
					PENABLE		RIGHT	LSB	CONTROL	IN	REQUIRED
					PWRITE		RIGHT	LSB	CONTROL	IN	REQUIRED
					PRDATA	32	RIGHT	LSB	DATA	OUT	REQUIRED
					AWID	4	RIGHT		CONTROL	OUT	REQUIRED

Once you fill in the details in the spreadsheet, import that spreadsheet in GenSys by using the following Tcl command:

```
import_data -file <filename.xml> -type excel
```

Using the Interfaces Suggested in GenSys

For details, see Inferring Interfaces Based on Suggestions from the Tool.

Adding Ports On Interfaces

You can add ports to an interface in any of the following ways:

- Using the Canvas View to Add Interface Ports
- Using the Tabular View to Add Interface Ports
- Using Tcl to Add Interface Ports

Using the Canvas View to Add Interface Ports

For details, see Adding Ports.

Using the Tabular View to Add Interface Ports

Perform the following steps to add interface ports in the tabular view:

1. Click the

2. button in the GUI to enter into the tabular view.
3. Double-click on the interface in the *Library Browser* tab to load that interface.
4. Click the *logical_ports* tab.
5. Click

6. to insert a new row in the table in the *logical_ports* tab.
7. Specify port details in the newly inserted row.

The following figure shows the ports information in the table in the *logical_ports* tab:

Info | Name | logical_ports | extendsTable | FileSets | attributes

	LogicalPortName	ActualPortName	Direction	lsb_bit	MSB	LSB
0	ARADDR		OUT	RIGHT	31	0
1	ARBURST		OUT	RIGHT	1	0
2	ARCACHE		OUT	RIGHT	3	0
3	ARID		OUT	RIGHT	3	0
4	ARLEN		OUT	RIGHT	3	0
5	ARLOCK		OUT	RIGHT	1	0
6	ARPROT		OUT	RIGHT	2	0
7	ARREADY		IN	RIGHT		
8	ARSIZE		OUT	RIGHT	2	0

Using Tcl to Add Interface Ports

Use the set_interface_port Tcl command to add interface ports.

For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

Creating Interface Instances

To create interface instances in GenSys GUI, drag and drop interfaces from Library Browser to a component or design subsystem under the Design Browser.

After instantiating an interface, you can specify its type as MASTER, SLAVE, SYSTEM, or MONITOR. By default, InterfaceLib_Type is set as MASTER.

Mapping Interfaces

Mapping interfaces means to identify interfaces on components and designs and mapping the actual ports of the component or design with the logical ports of the interface.

The following figure explains the mapping of interfaces and generating connections between the mapped interfaces:

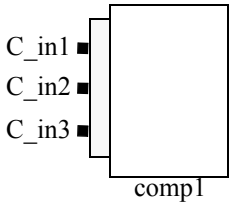
Interface Definition

The interface intf1 has three logical ports: in1, in2, and in3

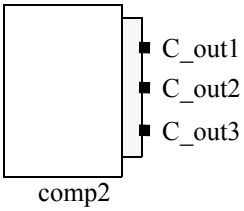
The interface intf2 has three logical ports: out1, out2, and out3

Mapping Interfaces Instantiated on Components:

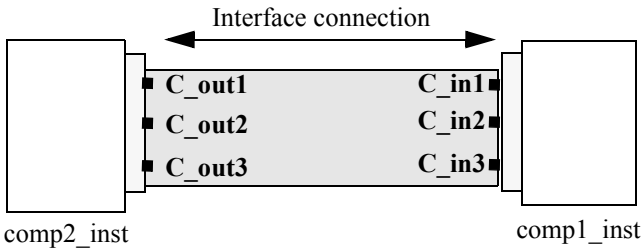
intf1 instantiated on the comp1 component,
and logical ports of intf1 are mapped with the actual ports of comp1:
in1 => C_in1
in2 => C_in2
in3 => C_in3



intf2 instantiated on the comp2 component,
and logical ports of intf2 are mapped with the actual ports of comp2:
ina => C_out1
inb => C_out2
inc => C_out3



Instances of comp1 and comp2 connected in the design des through one interface connection rather than three connections.



You can map interfaces in the following ways:

- Using the Interface Matching Dialog to Map Interfaces
- Using Tcl to Map Interfaces
- Using the Tabular View to Map Interfaces
- Using Rule-Based Interface Mapping

Advantages of Mapping Interfaces

The advantages are as follows:

- Mapping interfaces raises the level of design abstraction and makes design connectivity easier.

- It aids in design debug and analysis.

Using the Interface Matching Dialog to Map Interfaces

Use the Interface Matching dialog to map actual ports of a component with interface ports. Actual ports can be vector ports, scalar ports, and port bits.

In addition, you can map a single interface logical port to multiple physical or actual ports. Therefore, different slices of logical port are mapped to different physical ports. For example:

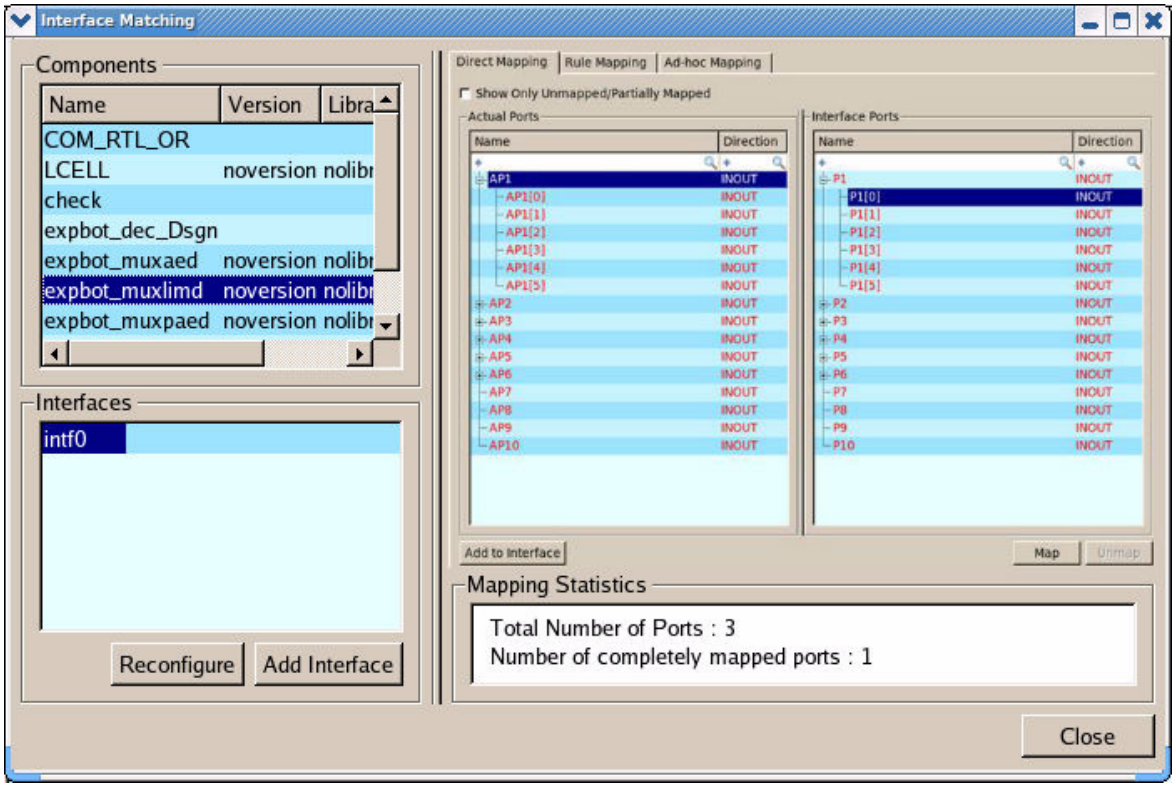
Slice 2:0 of Logical port L1 of interface I1 mapped to actual port P1

Slice 4:3 of Logical port L1 of interface I1 mapped to actual port P2 (1:0)

Slice 6:5 of Logical port L1 of interface I1 mapped to actual port P3(3:2)

Note:
The width of the logical slice must be the same as that of the physical port slice.

To open the Interface Matching dialog, click Tools > Interface Matching Widget. The following figure shows the Interface Matching dialog for a currently loaded design:



This dialog contains five sections:

- Components: This section lists the masters of all the instances that the are contained in the currently loaded design, including the top level design. The other sections are populated after you have selected a component. By default, a component is selected.
- Interfaces: This section lists the interfaces in the selected component.
- Actual Ports: This section lists the actual ports of the selected component.
- Interface Ports: This section lists the logical ports of the selected interface.
- Mapping Statistics: This section lists the mapping status.

All the mapped ports appear in green color and all the un-mapped ports appear in red color.

Note:
While the Interface Matching dialog is open, the rest of the GenSys GUI becomes read-only.

Performing Direct Mapping of Interface Ports

To directly map component actual ports with interface ports, perform the following steps in the Interface Matching dialog:

1. Select the required component from the Components section of the dialog.

   When you select a component:

   - All ports of that component appear in the Actual Ports section.
   - All interfaces instantiated in that component appear under the Interfaces section.
   - All logical ports of the currently selected interface in the Interfaces Section appear in Interfaces Ports section.

2. In the Actual Ports section, select the required port that you want to map with an interface port. You can multiple bits of a vector port and map to one interface port. Alternatively, you can map a single vector or scalar port to an interface port.
3. In the Interface Ports section, select the required interface port that you want to map with the already selected actual port of a component.
4. Click the Map button.

After performing the above steps, the selected ports are mapped. When you double-click an actual port, the mapped Interface Port is automatically selected (and vice-a-versa). This signifies the mapped ports.

To view only un-mapped ports in the above page, select the Show Only Unmapped/Partially Mapped option.

Un-mapping Ports

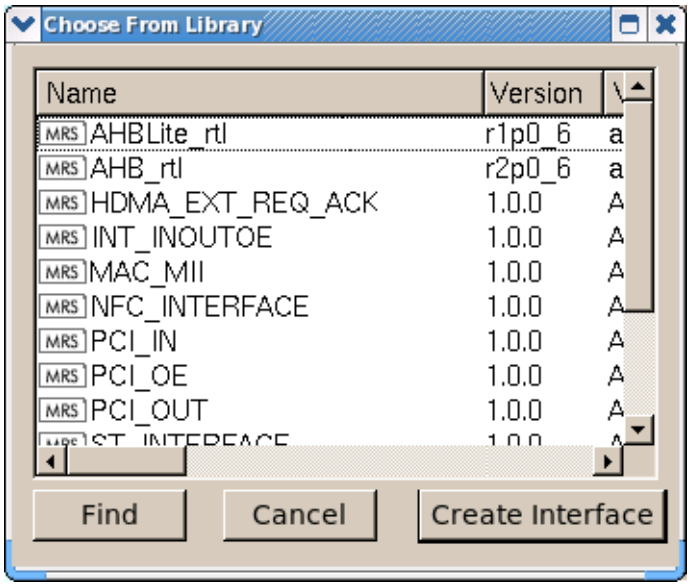
The Unmap button is enabled only if you have selected a logical port in the Interface Ports section. To un-map the ports, select the required logical ports in the Interface Ports section and click the Unmap button.

Adding Interface Instance through the Interface Matching Dialog

To add an interface instance of an interface displayed in the Library Browser Tab, perform the following steps:

1. Click the Add Interface button.

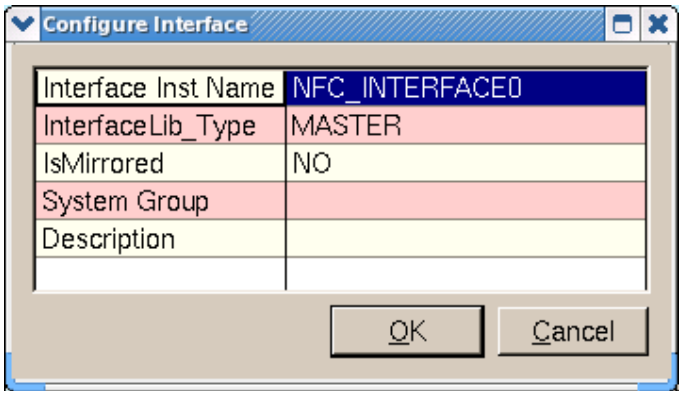
   This displays the Choose From Library dialog, as shown in the following figure:



   The above dialog shows all interfaces loaded in the Library Browser Tab.

2. Select the required interface from the above dialog.
3. Click the Create Interface button.

   The Configure Interface dialog appears, as shown in the following figure:



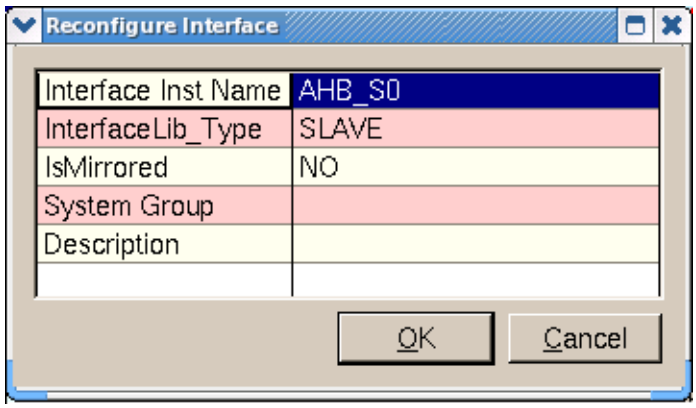
4. Specify the required details in the above dialog.
5. Click the OK button.

After performing the above steps, the specified interface instance gets added in the component selected in the Interface Matching dialog.

Reconfiguring Interfaces through the Interface Matching Dialog

To modify certain details of an interface displayed for a sub-system in the Interface Matching dialog, perform the following steps:

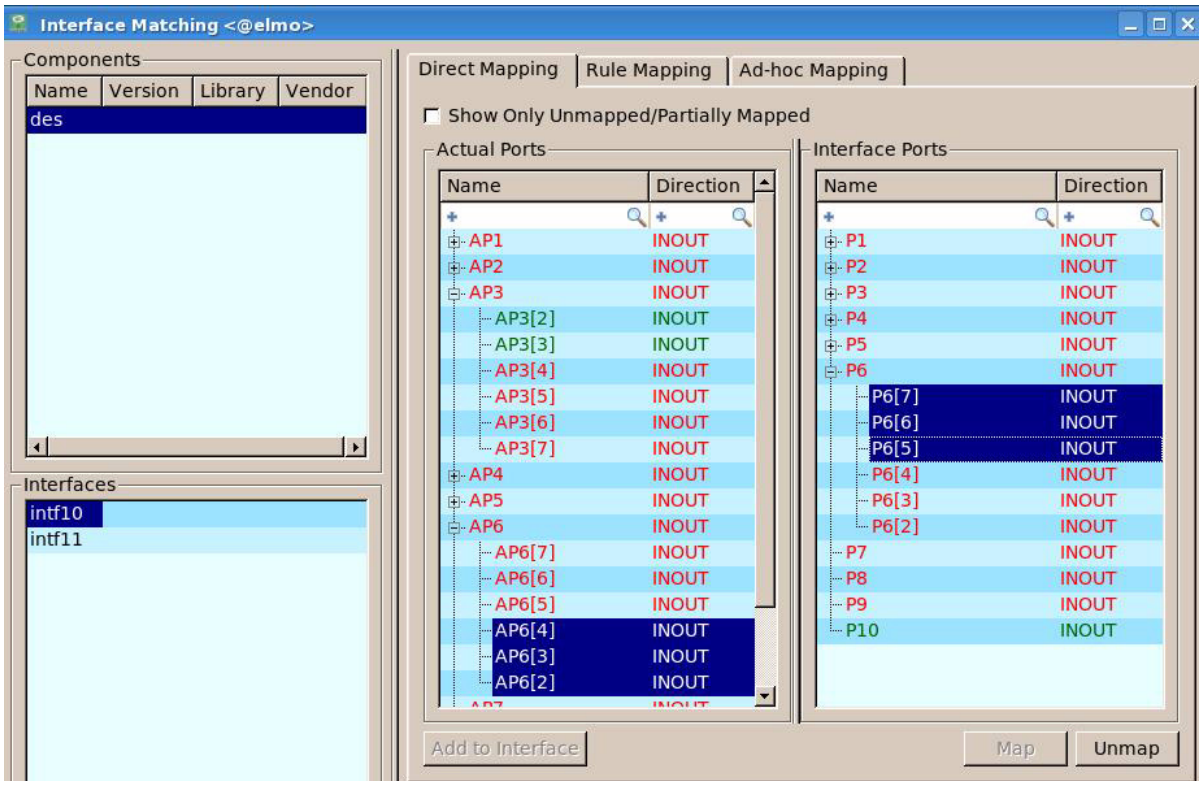
1. Select the required component in the Components section of this dialog.
2. Select the required interface in the Interfaces section.
3. Click the Reconfigure button.
4. The Reconfigure Interface dialog appears, as shown in the following figure:



5. Specify the required details in the above dialog.
6. Click the OK button to apply the specified changes.

Viewing Mapped Ports

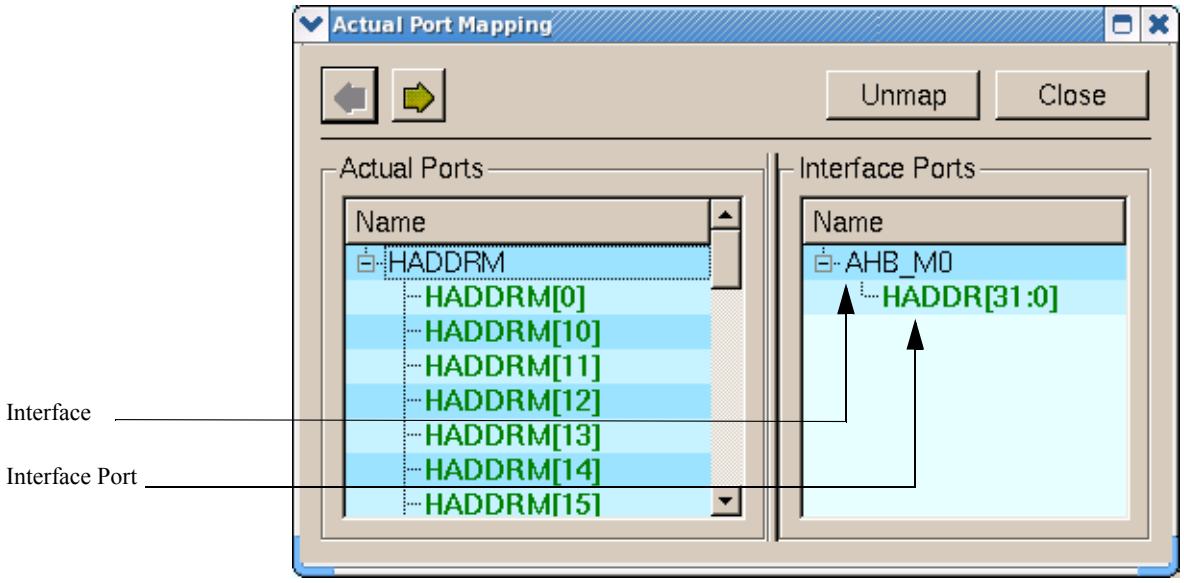
To view logical ports mapped with actual ports or vice-versa in the interface matching dialog, double-click on the required port. GenSys highlights the mapped ports in this dialog, as shown in the following figure:



In the above example, when the user clicks the P6[7] port bit in the Interface Ports section, GenSys selects the corresponding mapped ports in the Actual Ports section and the complete slice of the vector that was mapped in the Interface Ports. Therefore, in the above example, P6[7:5] and AP6[4:2] get selected when you double-click P6[7].

If you double-click an actual port that is mapped to multiple interface ports or an interface port that belongs to an interface that is not currently selected, the Actual Port Mapping dialog appears.

The following figure shows the Actual Port Mapping dialog:



Use the arrow buttons in the top left corner to navigate through the mappings. To un-map the selected mappings, click the Unmap button.

Using Tcl to Map Interfaces

You can use the set_interface_port or add_interface_mapping Tcl command to map interfaces.

Using the set_interface_port Tcl Command

In this case, perform the following steps to map interface logical ports with the component actual ports:

1. Create a component by using the new component <*comp-name*> Tcl command.
2. Add actual ports of the component by using the add_port Tcl command.
3. Instantiate an interface on the component by using the add_interface Tcl command.
4. Specify the mapping between the component actual to the interface logical port by using the set_interface_port Tcl command.

For details on these commands, refer to the *GenSys Tcl Commands Reference Guide*.

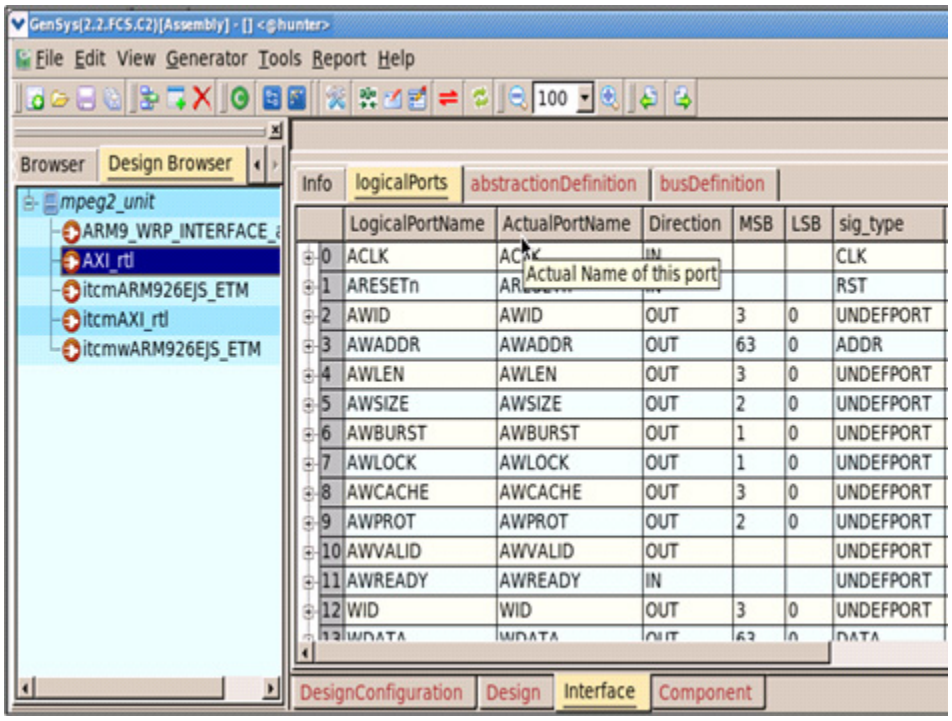
Using the add_interface_mapping Tcl Command

In this case, you can map interfaces in one go after creating multiple components and instantiating interfaces on them.

For details on the add_interface_mapping Tcl command, refer to the *GenSys Tcl Commands Reference Guide*.

Using the Tabular View to Map Interfaces

Click the tab and specify the mapping in the table under this tab, as shown in the following figure:



Using Rule-Based Interface Mapping

You can define rules to map actual ports of a component/design to the logical interface ports.

To do rule based interface matching, perform the following steps:

1. Use the make_rule Tcl command to define a pattern to be applied on a logical port name so that GenSys returns the matching actual port name.
2. Use the apply_rule Tcl command to apply rules defined by using the make_rule command on components and designs.

3. Use the match_interface Tcl command to specify the properties of interfaces that should be considered for mapping.
4. Use the infer_interfaces Tcl command to infer interfaces based on the defined rules.

Simple Pattern Matching Rule

Consider the following command:

```
make_rule -name Simple
```

Here, there are no substrings and no templates. This is an exact match rule. If the logical port name is portA, the actual port name should be portA as well. Or, if the -case_insensitive_match TRUE is used in the match_interface command, then even porta, PortA, PORTA will match.

Using Prefixes and Postfixes while Creating Rules

Consider the following command:

```
make_rule -name UsePrefix -prefix R_
```

The above command will match if the actual port name has R_ prefixed before the logical port name. For example, if a logical port name is DATAIN and the actual port name is R_DATAIN, there will be a match.

Postfixes operate similar to prefixes, except, of course at the end of the string.

Using Substrings while Creating Rules

Consider the following command:

```
make_rule -name UseSubstrings -substr {2:$}
```

the above command will select the substring from the third character to the end of the logical port name, and use that as a template for the actual port name. So if the logical port name is L_AHB_ADDR, and the physical port name is AHB_ADDR, then UseSubstrings will produce a match.

In such expressions, ^ is used for the start of a string and $ is used for the end.

Using Prefixes and Substrings Together while Creating Rules

Consider the following command:

```
make_rule -name UsePrefixAndSubstrings -prefix R_ -substr {0:2} -substr
{9:$}
```

When the above rule is applied over a logical port name, AHB_DATA_P0, it gives R_AHBP0, and if there is a physical port by that name, the match occurs. The two substrings are concatenated and then a prefix applied to get the match string.

Using Regular Expressions while Creating Rules

Consider the following command:

```
make_rule -name UseRegex1 -substr {0:1} -regexp {R_$1}
```

Regular expressions give the maximum power to the user. The above rule when applied over PI_DATA gives us R_PI. Here, $1 is the first substring matched.

Consider another command, as given below:

```
make_rule -name UseRegex2 -substr {0:1} -substr {3:4} -regexp
{P\$_$1$2[0-9]*}
```

When the above command is applied over LS_PI_DATA, $1 becomes LS, and $2 becomes PI. If an actual port name is P$_LSPI56, it will match during application of the above.

Matching Vectors and Scalars

A vector logical port matches only with a vector actual port. Therefore, you cannot match logical ports LS0, LS1, and LS2 with actual port, A[2:0].

The size of the logical vector should be greater than that of the actual port. Therefore, LS[5:0] matches with A[3:0] but not with A[8:0].

Such restrictions have to do only with the mapping performed by RTL import, and may not be the general restrictions on logical-actual mapping done in GenSys.

Applying Constraints on Port Direction During Interface Matching

Interfaces can be of type MASTER, SLAVE, or SYSTEM.

During interface matching, GenSys considers all these interface types in sequence depending upon the direction of the actual port, in order to instantiate an interface of the suitable type. However, you can control this by using the -type argument of the match_interface Tcl command.

If you specify -type master with the match_interface command, GenSys puts a constraint that the instantiated interface has to be a MASTER interface. This means the port directions of the actual ports must match the required directions for a MASTER interface. The -type option can accept SLAVE and SYSTEM options as well.

If you specify -type master -mirrorcheck with the match_interface command, GenSys puts a constraint that the instantiated interface should be a mirrored version of the interface being matched.

Selecting Rules for Interface Matching

Once you create rule by using the make_rule Tcl command, you need to apply these rules to interfaces by using the apply_rule Tcl command, as shown in the following example:

```
apply_rule -interface intf1 -rule R1 -rule R2
```

The above command means that rules R1 and R2 should be used to infer intf1 on all components.

Consider another example, as shown below:

```
apply_rule -interface intf1 -interface intf2 -rule R1 -rule R2
```

The above command means that both intf1 and intf2 will make use of the rules, R1 and R2, during the inference process on any component.

Now, consider you want that the inference of interface, intf1, on components, comp1 and comp2, can be done through rules, R1 and R2. But inference of intf1 on components, comp3 and comp4, should be done by using R3 and R4. This control is provided by the -component option of the apply_rule command, as illustrated by the two commands below.

```
apply_rule -component comp1 -component comp2 -interface intf1 -rule -R1
-rule R2
apply_rule -component comp3 -component comp4 -interface intf1-rule -R3
-rule R4
```

Selecting Candidate Components and Interfaces to Match

In the above sections, we have defined about rules. We have also associated the rules with the interfaces or interface-component pairs to specify which rules will spring into action during inference of those interfaces. However, mere specification of all these rules and associations is not sufficient information to the RTL import process. You need to do the following before the RTL import is effected:

- Select the pairs we you actually want to try matching in the next load_rtl command.
- Since this matching process may not be exact, you need to guide the process with thresholds and parameters in the event of a partial match. For example, if only 5 out of 10 logical ports are matched, it needs to be determined whether the interface be instantiated on the component or not.

This is implemented by using the match_interface command, as shown in the following example:

```
match_interface -component comp1 -component comp2 -interfaceintf1
-interface intf2
```

The above command specifies the import process to select the components, comp1 and comp2, and check if intf1 and intf2 can be identified on them. To understand that how does this relate to the apply_rule command, consider that in the context of the given match_interface command, the following rule associations are specified by using the apply_rule command:

```
apply_rule -component comp1 -interface intf1 -rule R1
apply_rule -component comp1 -interface intf1 -interface intf2-rule R1
-rule R2
apply_rule -component comp1 -comnponent comp2 -interfaceintf1 -rule R3
```

The above specified associations would kick in when whenever the command, load_rtl mydesign.v, is issued, where mydesign.v contains the instances of comp1 and comp2.

Suppose after the above load_rtl command above, the following command is also issued:

```
match_interface -component comp1 -component comp2 -component comp3
-interface intf1 -interface intf2
```

Here, the load_rtl command followed by the above command will try to identify intf1 and intf2 on comp3 as well, and relevant rule associations with comp3 will be included in the process.

The sequence of the apply_rule and match_interface commands matter. The match_interface command will include only those associations created by using the apply_rule command till that point. Thus, in the following sequence, the second rule association is not available during the import process:

```
apply_rule -component comp1 -component comp2 -interface intf1 -rule R1
match_interface -component comp1 -component comp2 -interface intf1
-interface intf2
apply_rule -component comp1-interface intf2 -rule R2
load_rtl mydesign.v
```

Specifying Multiple match_interface Commands

Multiple match_interface commands can be specified, as shown in the following example:

```
apply_rule -component comp1 -component comp2 -interface intf1-rule R1
match_interface -component comp1 -component comp2 -interface intf1
-interface intf2
match_interface -component comp3 -component comp4 -interface intf1
-interface intf2
load_rtl mydesign.v
```

The above command tries to match on comp1 to comp4.

Multiple instances of the same interface will be identified. Thus, once intf2 has been instantiated, the algorithm will continue to check for another instance of intf2 against the remaining ports, along with the other interfaces.

Specifying Multiple apply_rule Commands

Multiple apply_rule commands for a specific interface or interface/component combination is same as the rules being given on the same line. Consider the following example:

```
apply_rule -interface intf1 -rule R1
apply_rule -interface intf1 -rule R2
```

The above example is same as the following command:

```
apply_rule -interface intf1 -rule R1 -rule R2
```

The same applies for interface-component combinations as well. For example:

```
apply_rule -component comp1 -component comp2 -interface intf1-rule R1
apply_rule -component comp1 -component comp2 -interface intf1-rule R2
```

The above command is same as the following command:

```
apply_rule -component comp1 -component comp2 -interface intf1-rule R1
-rule R2
```

When using complex combinations like the following, it is important to be aware of what rules get actually selected:

```
apply_rule -interface intf1 -rule R1 -rule R2
apply_rule -component comp1 -component comp2 -interface intf1-rule R3
apply_rule -component comp1 -component comp3 -interface intf1-rule R1
-rule R4
apply_rule -interface intf1 -rule R2 -rule R1
<match-interface-command>
<load_rtl-command>
```

This means the following:

- When evaluating intf1 for comp1, use the rules, R1, R2, R3, and R4 (in that sequence).
- When evaluating intf1 for comp2, use the rules, R1, R2, and R3 (in that sequence).
- When evaluating intf1 for comp3, use the rules R1, R2, and R4 (in that sequence).
- When evaluating intf1 for comp4, use the rules R1 and R2 (in that sequence).

Significance of the Order of Rules in the apply_rule Command

For a specific interface, the matching is done by using as many rules as specified in the apply_rule command. However, the order in which the rules are specified will determine the actual matches made between the logical and actual ports. Consider the following example:

RTL to be matched:

```
module Simple (AH1, AH2, AH3, AH4, OUT2, ABH1, ABH2, ABH3,ABH4, ABHOUT)
;input   AH1;input   AH2 ;input   AH3;input   AH4;output  OUT2;
input   ABH1 ;input   ABH2 ;input   ABH3;input   ABH4;output  ABHOUT;
endmodule
```

Following is the set of Tcl commands for interface definition:

```
new interface I1set_interface_port  -name H1 -direction
INset_interface_port  -name H2 -direction INset_interface_port  -name H3
-direction INset_interface_port  -name H4 -direction INset_interface_port
-name HOUT -direction OUTsave I1
```

Therefore, the interface has four input ports and a single output port.

Looking at the set of actual ports, the expectation is that the first five ports should match an instance of I1. Similarly, the last five ports should match another instance of I1. That is, the desired mapping is as follows:

Interface Instance	Logical Port	Actual Port
I1_inst1	H1	AH1
I1_inst1	H2	AH2
I1_inst1	H3	AH3
I1_inst1	H4	AH4
I1_inst1	HOUT	OUT2
I1_inst2	H1	ABH1
I1_inst2	H2	ABH2
I1_inst2	H3	ABH3
I1_inst2	H4	ABH4
I1_inst2	HOUT	ABHOUT

For this to happen, you need to create rules that match as many ports as possible. Since all ports have A or AB as prefix, you zoom onto the following rules:

```
make_rule -name R1 -substr {^:$} -regexp {A$1}make_rule -name R2 -substr
{^:$} -regexp {AB$1}
```

Subsequently, specify the following commands to effect the RTL import:

```
apply_rule -component Simple -interface I1 -rule R1 -rule R2
match_interface -component Simple -interface I1-case_insensitive_match
TRUE
load_rtl -component simple.v
```

Needless to say, simple.v contains the RTL code for the module, Simple, mentioned earlier.

Examining the match performed, we come across the following situation:

Interface Instance	Logical Port	Actual Port
I1_inst1	H1	AH1

Interface Instance	Logical Port	Actual Port
I1_inst1	H2	AH2
I1_inst1	H3	AH3
I1_inst1	H4	AH4
I1_inst1	HOUT	ABHOUT
I1_inst2	H1	ABH1
I1_inst2	H2	ABH2
I1_inst2	H3	ABH3
I1_inst2	H4	ABH4
I1_inst2	HOUT	<no mapping>

In the above case, as you can see, HOUT of I1_inst1 get mapped to ABHOUT instead of I1_inst2. Here, since R1 was specified first in the apply_rule command, a match takes place between actual ports AH1 to AH4 and logical H1 to H4 according to R1. However, there is no AHOUT in the port list. Therefore, GenSys picks up R2 to see if a match can be made. There is a match, and the port ABHOUT is mapped to logical HOUT.

Having instantiated I1_inst1, GenSys now tries mapping R1 and R2 again for the remaining ports. It finds a match for ABH1 to ABH4 by using the R2 rule, and creates another interface instance accordingly.

This, however, is not desired. Let us now reverse the rules in the apply_rule command, as shown below:

```
apply_rule -component Simple -interface I1 -rule R2 -rule R1
match_interface -component Simple -interface I1-case_insensitive_match
TRUE
load_rtl -component simple.v
```

Now, we get the following:

Interface Instance	Logical Port	Actual Port
I1_inst1	H1	ABH1

Interface Instance	Logical Port	Actual Port
I1_inst1	H2	ABH2
I1_inst1	H3	ABH3
I1_inst1	H4	ABH4
I1_inst1	HOUT	ABHOUT
I1_inst2	H1	AH1
I1_inst2	H2	AH2
I1_inst2	H3	AH3
I1_inst2	H4	AH4
I1_inst2	HOUT	<no mapping>

This is much better, leaving us to manually map logical port HOUT of I1_inst2 to OUT2 by using the set_interface_port command, as shown below:

```
set_interface_port  -name HOUT -actual_name OUT2-instance I1_inst2
```

Guiding the Performance of the Match Algorithm

An interface match should ideally map all the logical ports to its actual equivalents. This can happen through a few simple rules if the naming is consistent. However, often, it is not so, particularly the actual port names. Maybe the RTL was generated or has IPs from several quarters. Whatever it be, it is reasonable to expect that all these rules produce only a partial match of an interface. Then the question is whether to instantiate the interface or not. There are two guiding principles here:

- All the required ports of a logical interface must match.
- This is a gating principle. An AHB interface cannot be instantiated if some of the required ones such as data bus or address bus did not find a match.
- The user can guide the match performance by choosing either greedy or the bestfit parameter.

Bestfit

Here, the match process tries to match each interface on the component, and stores the total number of ports matched. After finishing off with all the interfaces, the interface with the highest number of ports matched is instantiated.

While matching by using the different rules, suppose there are multiple interface matches. Let us say, of the 10 ports of intf1, 9 match, whereas out of 20 ports of intf2, 12 match. With bestfit in force, intf2 is instantiated.

One may argue that a percentage was a better criterion. But remember that all the required ports have to match anyway. Thus those that do not match are the optional ports. In that case, it makes sense to instantiate the interface with the larger ports.

Once the decision is taken, the algorithm is run once again on the remaining ports till we reach a stage of no matches.

Greedy

Here, the user provides a threshold to decide whether to instantiate the interface or not. Decision is not postponed till all the interfaces are evaluated.

So in the case discussed in the previous section, suppose intf1 is matched first with 9 ports out of 10. If the threshold percentage is 0.9, intf1 is instantiated.

Specifying the Name of the Instantiated Interface

If intf1 is decided to be instantiated, it needs to be determined that what should be the name of the instantiated interface, that is intf10 or P_intf1. The user has full control over that by specifying the rule by using the -interface_naming option of the match_interface command. The rule specified will be through the standard make_rule command. Therefore, substrings of the logical port name and prefixes or regular expressions will be used to create a string that will be used to name the interface. For example:

```
make_rule -name MyInterfaceName -substr 0:1 -prefix P0
```

When applied over L_D1, this elicits L_P0.

Note:

This does not allow sequential naming of interfaces, such as L_P0, L_P1, etc. In fact, if there is a naming conflict, GenSys uses some internal principle to rename the last name in conflict.

Handling Case Sensitivity During Interface Matching

The -case_insensitive_match option of the match_interface command gives control over the case sensitivity.

The Significance of the match_interface Command

As you have seen so far, the match_interface command gives control over the following:

- Which components to match against what interfaces
- Algorithm to guide interface instantiation
- Naming of the interface instantiated
- Case sensitivity

Since multiple match_interface commands can be issued before a load_rtl command, this gives precise control over the interface matching process. for example:

```
match_interface -component comp1 -component comp2 -interface intf1
-interface intf2 -greedy 0.7 -interface_naming name1
-case_insensitive_match TRUE
match_interface -component comp3 -component comp4 -interface intf1
-interface intf2 -bestfit -interface_naming name2
-case_insensitive_match FALSE
load_rtl mydesign.v
```

Therefore, this accomplishes to identify intf1 and intf2 on components, comp1 to comp4, but the names of the instantiated interfaces, the algorithm to identify them, and the case considerations are quite different for comp1 and comp2 vs. comp3 and comp4.

Example of rule-Based Interface Matching

Consider that there are a set of actual ports and corresponding logical port names as follows:

Actual Port	Logical Port
Xyz	CLK
Abc	RST
M1_pread	PREAD
M1_pwrite	PWRITE
M1_OK	OK

Now consider an interface, intf1, comprised of three logical ports, PREAD, PWRITE, and OK. Then, in the above set of ports, the instance of the intf1 interface can be identified. The rest are random mappings.

Ports starting with M1 can be mapped with the following rule:

```
make_rule -substr {^:$} -regexp {M[0-9]_$1}
```

With a case insensitive search effected by using the match_interface command, there is an exact match for the three logical ports of the interface with M1.

For the other two ports, Xyz and Abc, the actual port names cannot be derived from the logical port names in any way. Therefore, one needs to use standard set_interface_port Tcl commands to map them to the respective logical ports.

Important Guidelines for Defining Interfaces

While deciding how you want to define interfaces for a design architecture, consider the following aspects of representation, since each interface definition can represent just one group of signals with fixed signal directions and widths:

- Interface representation: Determine if you want to represent interfaces as direct or mirrored (in the IP-XACT sense).

  A direct interface connection has one end as a master and the other end as a slave, irrespective of what they are connecting to. A mirrored connection has the “true” master end of a master connection, such as. a CPU as a master, and the bus-connection end as a mirrored-master. Similarly, a mirrored connection has the “true” slave end of a slave connection, such as a peripheral as a slave, and the bus-connection end as a mirrored-slave. These concepts can be represented in GenSys by creating an interface definition for each object. For example, for AHB:

  - For direct connection, create an AHB_M definition and an AHB_S definition
  - For mirrored connections, create AHB_M, AHB_MM and AHB_S, AHB_SS definitions

  You are not restricted to these names, but a similar naming convention will help ease-of-use in chip-assembly.

- The primary difference between these objects will be in port directions, although in the mirrored case you may possibly have different signals in the master and slave interfaces.
- For interface representation with different data/address widths, you should create new interface definitions for these cases. For a direct interface for example, you might create, AHB_32_M, AHB_32_S, AHB_64_M, AHB_64_Sand so on.
- While bus standards include clock and reset signals in the definition, we do not recommend these be included in an interface definition. Clock and reset signals are handled quite differently in hookup than other signals in an interface. While it is possible and legal to include them in the definition, you will generally find you do not gain any advantage by including them in GenSys interface definitions. However, GenSys does provide a method to associate clock and reset ports on a component with an interface on that component.

You do not need to create new interface definitions to handle different clock speeds or resets. As mentioned above, these are handled outside the interface.

Building Components

Building components includes creating components, parameterizing them, and instantiating interfaces on them to create connections.

You can also import existing components through RTL import.

Creating Components

You can create components in any of the following ways:

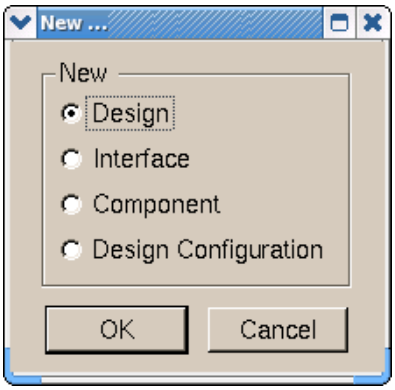
- Using the Canvas View to Create Components
- Using the Tabular View to Create Components
- Using Tcl to Create Components
- Using MS Excel to Create Components

Using the Canvas View to Create Components

1. Click

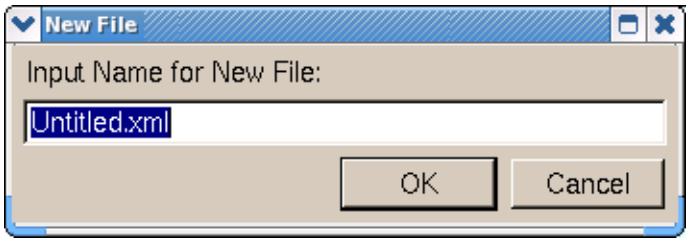
2. icon in the toolbar. Alternatively, select the File->New menu option.

   The New dialog appears, as shown in the following figure:



3. Select the Component option in the above dialog and click the OK button.

   The New File dialog appears, as shown in the following figure:



4. Specify the name of the new component in the above dialog, and click the OK button.

   Consider that you specify the name as IP926EJS. The new component appears in the schematic of the Canvas View, as shown in the following figure:

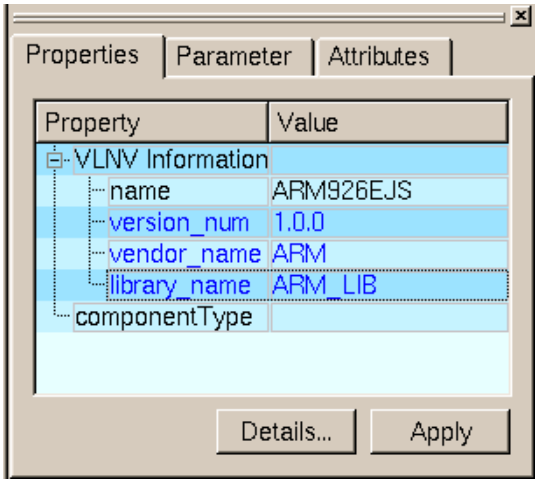
5. Add VLV information for this component by performing the following steps:

   a. Enter in the edit mode by clicking

   b. from the schematic toolbar of the Canvas View.

   c. Go to the Property Editor pane and click the Properties tab of this pane.

   d. Enter VLV information under the Properties tab, as shown in the following figure:



e. Click the Apply button to apply the VLV information on the component.

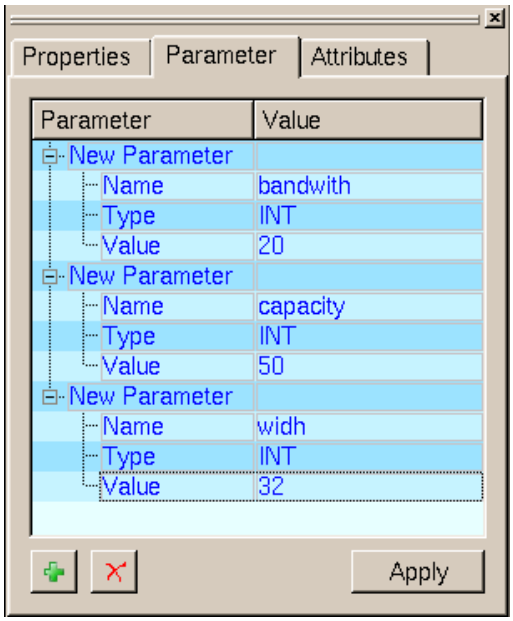
6. Add parameters to the component by performing the following steps:

a. Click the Parameters tab of the Property Editor pane.

b. Click

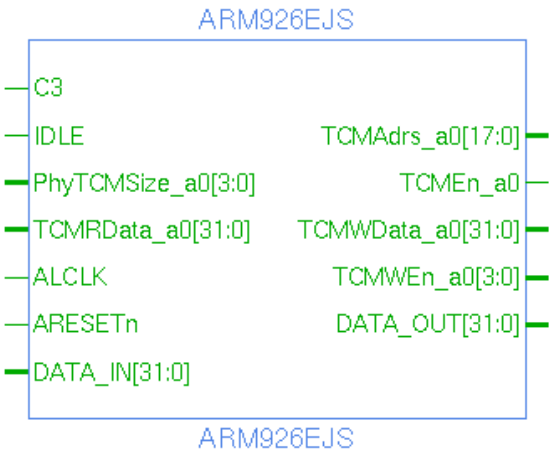
c. to add rows under this tab.

d. Specify parameter information in the newly added rows, as shown in the following figure:



7. Add ports to the selected component. For details, see Adding Ports.

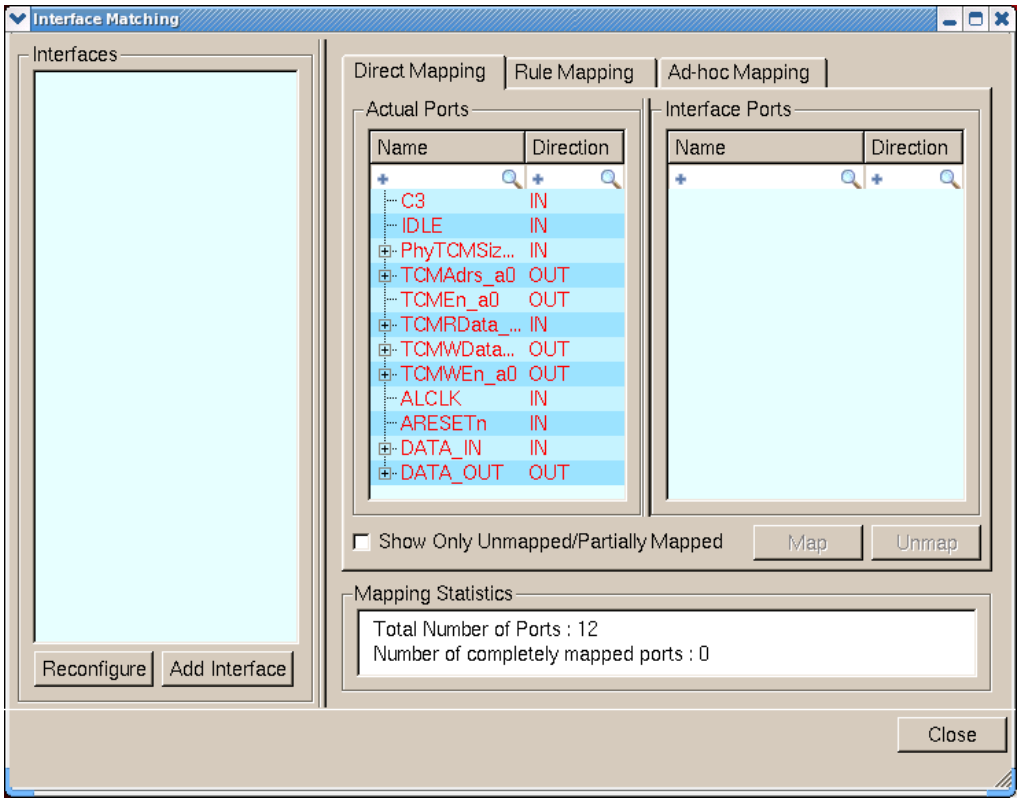
   Consider that you have added ports so that the component now appears as shown below:



8. Instantiate an interface from the library and map the ports of the component (actual ports) to the ports of the interface (logical ports) by performing the following steps:

a. Click

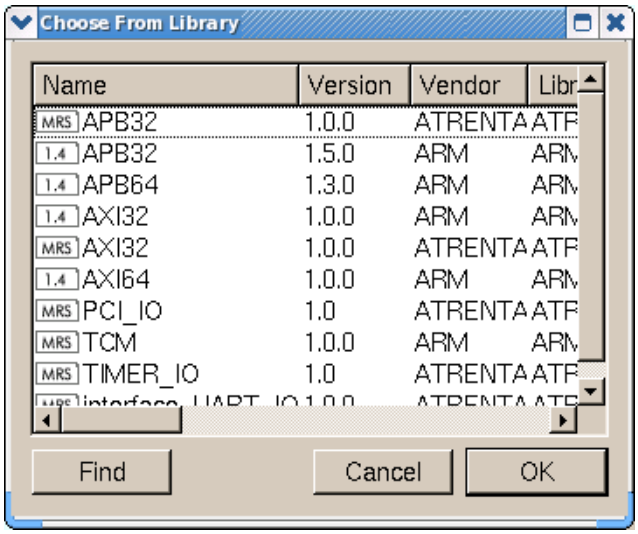
b. from the toolbar. The Interface Matching Widget dialog appears, as shown in the following figure:



For details on the above dialog, see Mapping Interfaces.

a. Click the Add Interface button in the above dialog.

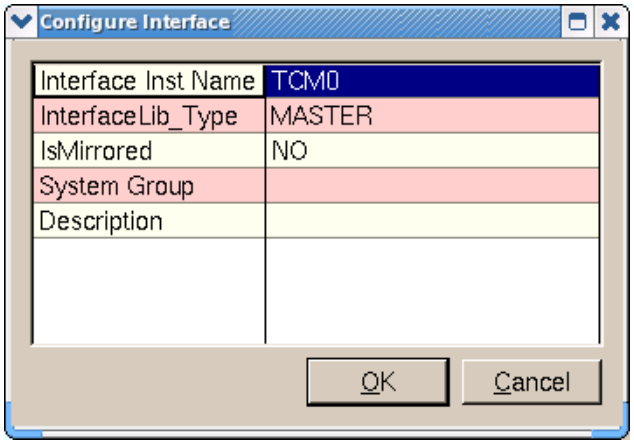
The following dialog appears:



The above dialog lists interfaces available in the library path.

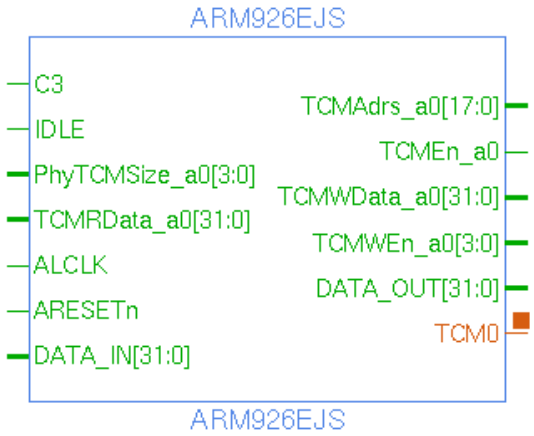
a. To configure any interface (say TCM), select that interface from the above dialog and click the OK button.

The Configure Interface dialog appears, as shown in the following figure:



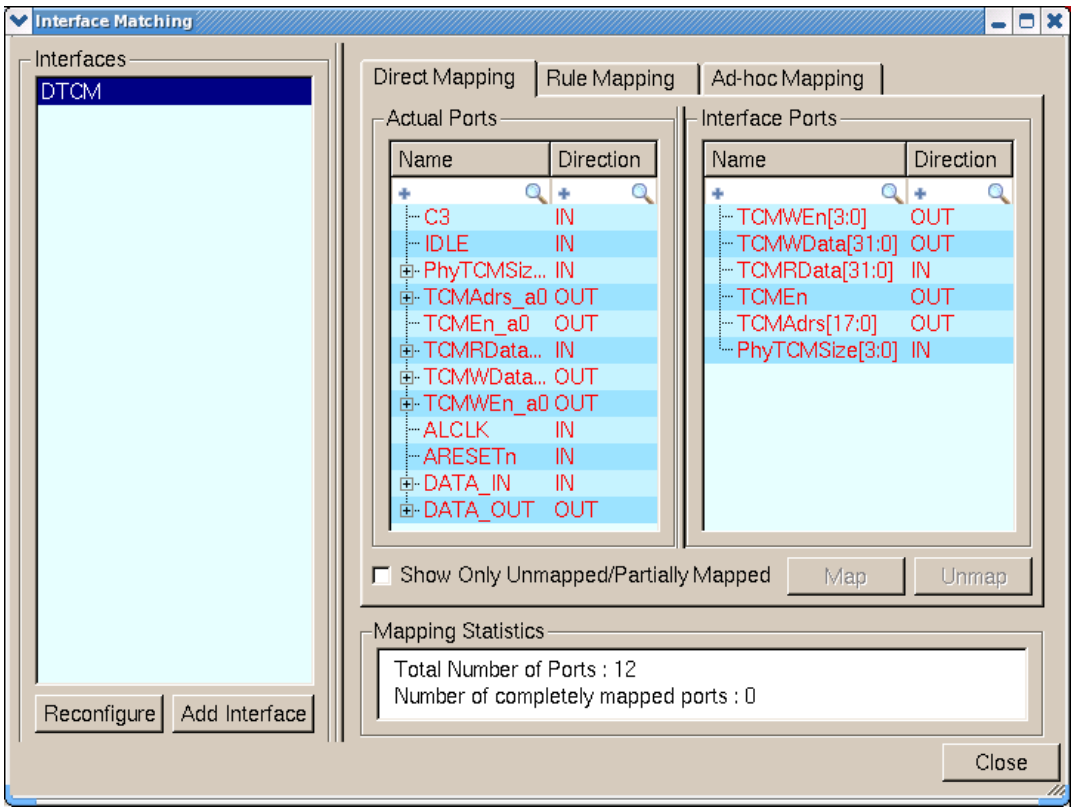
a. In the above dialog, modify the details as per your requirement and click the OK button. For example, modify the interface instance name to DTCM.

After performing this step, an instance of the TCM interface is added to the component, as shown in the following figure:



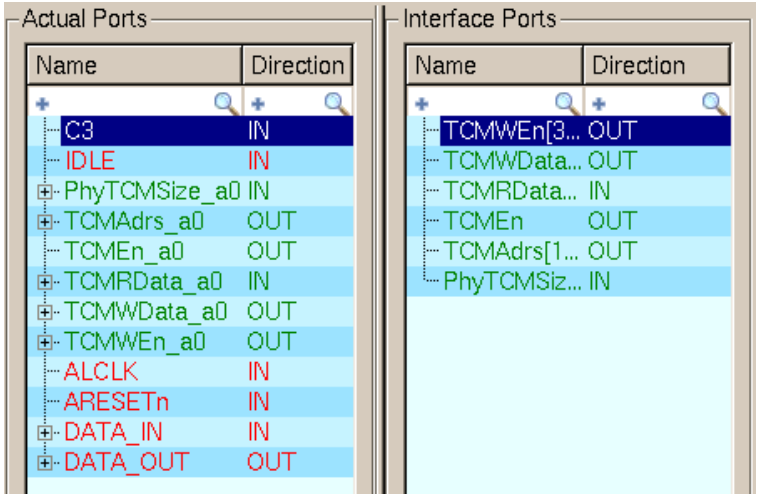
In addition, the DTCM instance appears under the Interfaces section of the Interface Matching Widget dialog, the component ports appear in the Actual Ports section, and the logical ports of the DTCM interface instance appear in the Interface Ports pane.

This is shown in the following figure:

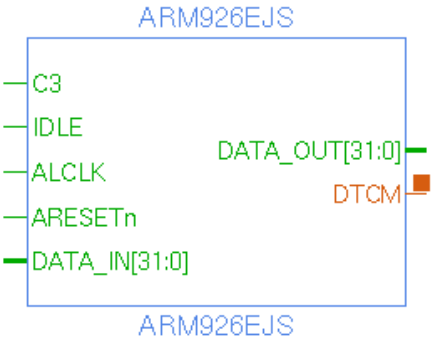


a. To perform actual to logical port mapping, select a logical port and an actual port and click on the Map button.

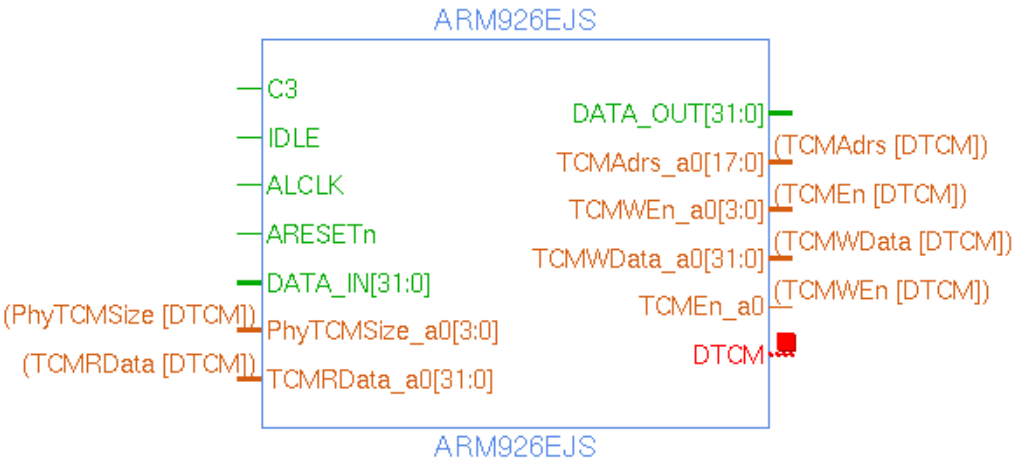
All the mapped interfaces appear in green color, as shown below:



After performing the above steps, refresh the schematic in the Canvas View. The component now appears as shown below:



In the above view, to expand the ports of the DTCM interface to see how the ports have been mapped to this interface, right-click on this interface and select the Expand Interface Ports option from the shortcut menu. The following figure shows the schematic of the component displaying expanded interface ports:



To collapse the expanded ports, right-click on the component and select the Collapse Interface Ports option from the shortcut menu.

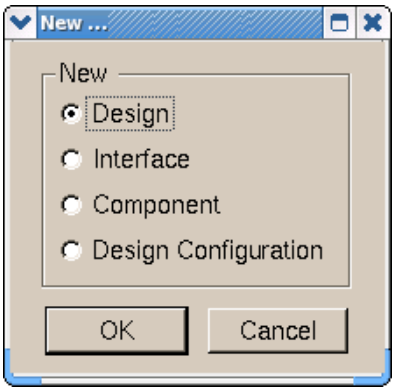
Using the Tabular View to Create Components

To create components in the tabular view, perform the following steps:

1. Click

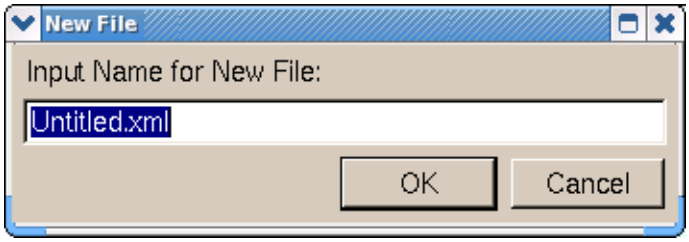
2. button to switch to the tabular view.
3. Select the *File-> New* menu option.

   The New dialog appears, as shown in the following figure:



4. Select the Component option in the above dialog and click the OK button.

   The New File dialog appears, as shown in the following figure:



5. Specify the name of the new component in the above dialog, and click the OK button.

   The component appears under the *Design Browser* tab.

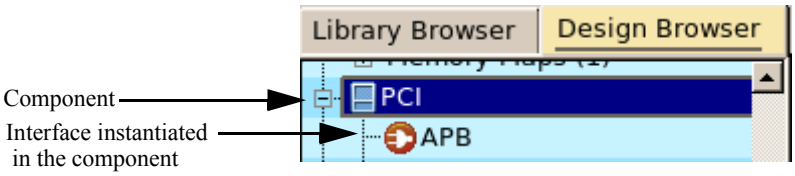
6. Specify the component details in the tables under different tabs, such as:
   - The *Info* tab: Specify information, such as the component type and description.
   - The *Name* tab: Specify the VLNV information.
   - The *Ports* tab: Specify the ports information, as shown in the following figure:

Info | Name | Ports | Parameters | attributes

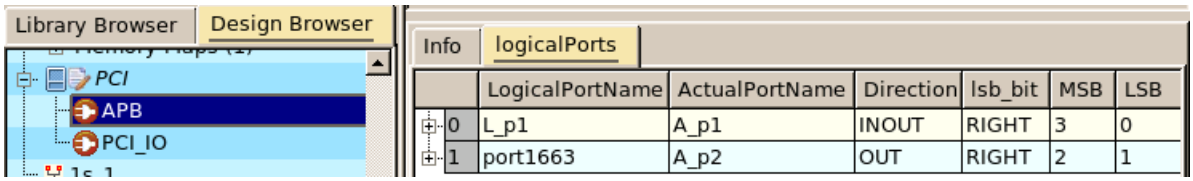
	Name	direction	MSB	LSB	port_type	align	lsb_bit	Interfaces
0	gpio_i	IN	111	0	CORE_OUTPUT	MSB	RIGHT	[111:0]:GPIO..
1	gpio_o	OUT	111	0	CORE_INPUT	MSB	RIGHT	[111:0]:GPIO..
2	gpio_oe_n	OUT	111	0	IO_OEN	MSB	RIGHT	[111:0]:GPIO..
3	gpio_psel	OUT	111	0	CORE_INPUT	MSB	RIGHT	[111:0]:GPIO..
4	gpio_pdis	OUT	111	0	CORE_INPUT	MSB	RIGHT	[111:0]:GPIO..
5	PCLK	IN			CLK	MSB	RIGHT	
6	PRESETn	IN			RST	MSB	RIGHT	
7	IDLE	OUT			CONTROL	MSB	RIGHT	
8	cfg	IN	3	0		MSB	RIGHT	
9	PWR_CTL	IN	3	0		MSB	RIGHT	

7. Instantiate interfaces in the component by performing the following steps:
   - Click the Library Browser tab.
   - Select the interface to be instantiated and press the Ctrl+C keyboard keys.
   - Click the Design Browser tab.
   - Select the component and press the Ctrl+V keyboard keys.

   After performing the above steps, the instance of the selected interface appears under the component hierarchy, as shown in the following figure:



8. Double-click on the added interface instance to load the tables of this instance in the tabular view.
9. Click the *logicalPorts* tab.
10. Specify the logical to actual ports mapping in the table under the *logicalPorts* tab, as shown in the following figure:



You can add more component details by specifying information under different tabs.

Using Tcl to Create Components

Use the following Tcl commands to create a component and specify ports and interface information:

- new component <component-name>: Use this command to create a component.
- add_port: Use this command to add ports on the component.
- add_interface: Use this command to add an interface on the component.
- set_interface_port: Use this command to map logical ports of the interface with the actual ports of the component.
- save: Use this command to save the component.

For details on the above commands, refer to the *GenSys Tcl Commands Reference Guide*.

Example

The following example uses the above commands to create the ip1156 component and specify interface details on it:

```
new component ip1156 -vendor_name {IP} -library_name {X3570} -version_num
{1.0.0}
add_interface -name AXI -interface_ref {AXI64} -vendor {IP} -library
{X3570} -version {1.0.0} -type {MASTER}
set_interface_port -name ARADDR   -actual_name ARADDR_a0  -instance AXI
-msb {63} -lsb {0}
set_interface_port -name ARBURST  -actual_name ARBURST_a0 -instance AXI
-msb {1} -lsb {0}
set_interface_port -name ARCACHE  -actual_name ARCACHE_a0 -instance AXI
-msb {3} -lsb {0}
set_interface_port -name ARID     -actual_name ARID_a0    -instance AXI
-msb {3} -lsb {0}
set_interface_port -name ARLEN    -actual_name ARLEN_a0   -instance AXI
-msb {3} -lsb {0}
set_interface_port -name ARLOCK   -actual_name ARLOCK_a0  -instance AXI
-msb {1} -lsb {0}
set_interface_port -name ARPROT   -actual_name ARPROT_a0  -instance AXI
-msb {2} -lsb {0}
```

```
set_interface_port -name ARREADY  -actual_name ARREADY_a0 -instance AXI
set_interface_port -name ARSIZE   -actual_name ARSIZE_a0  -instance AXI
-msb {2} -lsb {0}
set_interface_port -name ARVALID  -actual_name ARVALID_a0 -instance AXI
save ip1156
```

Using MS Excel to Create Components

You can create a component by specifying its details in an MS Excel spreadsheet (XML SpreadSheet 2003), and then importing that sheet in GenSys.

The following figure shows such spreadsheet:

A	B	C	D	E	F	G	H	L	M	N	O	P	Q	R	S
	VLNV			Info											
Vendor	Library	Name	Version	Type	RTL Library	RTL Package	Description								
ARM	ARM_LIB	arm1156	1.0.0	CPU											
Ports								Interface							
Name	Direction	MSB	LSB	LSB_Bit	Align	Signal Type	Required	Vendor	Library	Name	Version	Instance	Mirror	Master/Slave	Logical Port
TCMAdrs_a0	OUT	17	0	RIGHT	MSB	ADDR	REQUIRED	ARM	ARM_LIB	TCM	1.0.0	DTCM			TCMAdrs
ITCMAdrs	OUT	17	0	RIGHT	MSB	ADDR	REQUIRED			TCM		ITCM			TCMAdrs
ARADDR_a0	OUT	63	0	RIGHT	MSB	ADDR	REQUIRED	ARM	ARM_LIB	AXI64	1.0.0	AXI			ARADDR
AWADDR_a0	OUT	63	0	RIGHT	MSB	ADDR	REQUIRED								AWADDR
PhyTCMSize	IN	3	0	RIGHT	MSB	DATA	REQUIRED	ARM	ARM_LIB	TCM	1.0.0	ITCM			PhyTCMSize
VDATA_a0	OUT	63	0	RIGHT	MSB	DATA	REQUIRED	ARM	ARM_LIB	AXI64	1.0.0	AXI			VDATA
RDATA_a0	IN	63	0	RIGHT	MSB	DATA	REQUIRED								RDATA
TCMWData_a0	OUT	31	0	RIGHT	MSB	DATA	REQUIRED	ARM	ARM_LIB	TCM	1.0.0	DTCM			TCMWData
TCMRData_a0	IN	31	0	RIGHT	MSB	DATA	REQUIRED								TCMRData
PhyTCMSize_a0	IN	3	0	RIGHT	MSB	DATA	REQUIRED								PhyTCMSize
ITCMWData	OUT	31	0	RIGHT	MSB	DATA	REQUIRED					ITCM			TCMWData
ITCMRData	IN	31	0	RIGHT	MSB	DATA	REQUIRED								TCMRData
TCMEn_a0	OUT			RIGHT	MSB	CONTROL	REQUIRED					DTCM			TCMEn
TCMWEn_a0	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED								TCMWEn
ITCMEn	OUT			RIGHT	MSB	CONTROL	REQUIRED					ITCM			TCMEn
ITCMWEn	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED								TCMWEn
AWID_a0	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED								AWID
AWLEN_a0	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED								AWLEN
AWSIZE_a0	OUT	2	0	RIGHT	MSB	CONTROL	REQUIRED								AWSIZE
AWBURST_a0	OUT	1	0	RIGHT	MSB	CONTROL	REQUIRED								AWBURST
AWLOCK_a0	OUT	1	0	RIGHT	MSB	CONTROL	REQUIRED								AWLOCK
AWCACHE_a0	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED								AWCACHE
AWPROT_a0	OUT	2	0	RIGHT	MSB	CONTROL	REQUIRED								AWPROT
AWVALID_a0	OUT			RIGHT	MSB	CONTROL	REQUIRED								AWVALID
WID_a0	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED								WID
WSTRB_a0	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED						NO	master	WSTRB
WLAST_a0	OUT			RIGHT	MSB	CONTROL	REQUIRED								WLAST
WVALID_a0	OUT			RIGHT	MSB	CONTROL	REQUIRED								WVALID
BREADY_a0	OUT			RIGHT	MSB	CONTROL	REQUIRED								BREADY
ARID_a0	OUT	3	0	RIGHT	MSB	CONTROL	REQUIRED								ARID

Once you fill in the details in the spreadsheet, import that spreadsheet in GenSys by using the following Tcl command:

```
import_data -file <filename.xml> -type excel
```

Importing Components

You can import components in GenSys by using the *File -> Load RTL* menu option or by using the load_rtl Tcl command. For details on RTL import, see Performing RTL Import.

When you import a component:

- It has the same name as the RTL module name as well as the same actual port names as present in the RTL.
- Any parameters/generics that are part of the RTL are preserved as parameters in GenSys.

Handling VHDL Designs During RTL Import

If a VHDL design contains package definitions, you must precompile the packages followed by mapping logical to physical libraries before reading in the RTL.

Handling SystemVerilog IPs During RTL Import

While importing SystemVerilog IPs, specify the -enableSV command during RTL import, as shown in the following example:

```
load_rtl sys_verilog.sv -enableSV
```

Mapping Interfaces on Components

For details, see Mapping Interfaces.

Building Designs

You can create designs hierarchically or at the flat level.

For details on building designs, see Building SoCs.

Working with Attributes and Parameters

Attributes and parameters are name/value pairs that are used to defined certain user-defined properties for an object.

Working with Attributes

Attributes are name/value pairs of predefined types, such as BOOL, INTEGER, STRING, and ENUM.

They are commonly used to define values that are important to generators. For example:

- You can define the gate_count attribute on a component to estimate the gate count to be used in early floor-planning estimation.

- You can define the interface_type attribute on an interface to specify a type, such as IO, which could be used by an IO generator.

Assigning Attributes to Objects

You can assign attributes to an object in any of the following ways:

- Using the Canvas View to Assign Attribute
- Using the Tabular View to Assign Attribute
- Using Tcl to Assign Attribute

Using the Canvas View to Assign Attribute

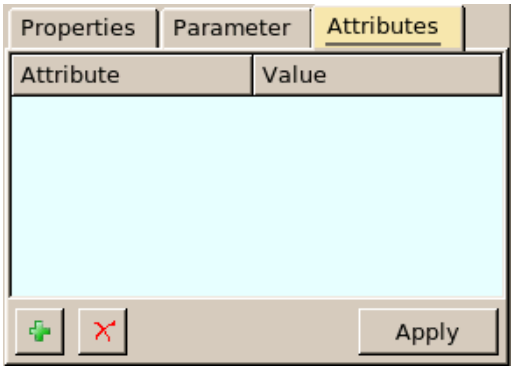
To assign an attribute to an object in the Canvas View, perform the following steps:

1. Click

2. in the GUI window to enter in the Canvas View.
3. Double-click on the object in the *Library Browser*.

   The object is loaded in the *Design Browser*.
4. Click the *Attributes* tab in the *Property Editor* pane.

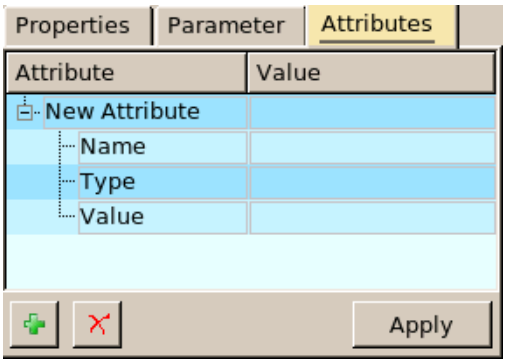
   The following figure shows the area under the *Attributes* tab:



5. In the above page, click

6. to add a row for specifying attribute details.

   New rows appear, as shown in the following figure:



7. Specify attribute details, such as name, type, and value in the above page.
8. Repeat step 4 to add more rows.
9. Click the *Apply* button to assign the attributes to the selected object.

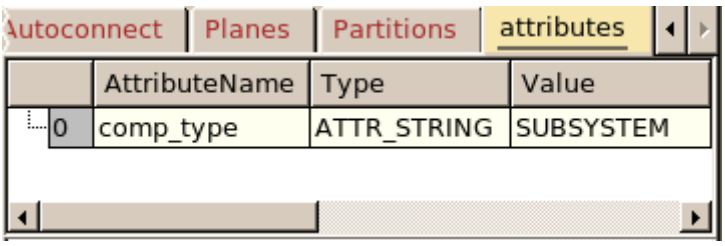
Using the Tabular View to Assign Attribute

To assign an attribute to an object in the Canvas View, perform the following steps:

1. Click

2. in the GUI window to enter in the tabular view.
3. Double-click on the object in the *Library Browser*.

   The object is loaded in the *Design Browser*.
4. Click the *Attributes* tab.
5. Specify attribute details, such as name, type, and value, as shown in the following figure:



Using Tcl to Assign Attribute

Use the add_attribute Tcl command to assign an attribute to an object.

Run this command immediately after creating the object or after select the object.

Retrieving Attribute Values

You can retrieve the attribute value through an expression (in a table entry or Tcl argument) by using the formula [a <name>].

For example, the entry [a attribute_A] would return the value of the attribute, attribute_A, on the current object.

Working with Parameters

Parameters are name-value pairs of predefined GenSys types, such as BOOL, STRING, INTEGER, and ENUM.

Note:
Parameters are applicable to only designs and components.

Parameters are used in RTL of objects. For example:

- You can create the width parameter on a component to define the width of a bus. The width can be overridden at the component instance level.
- You can create the arb_type parameter for a bus component to define the arbitration type, such as fixed or round_robin.

  This is the example of a parameter that is not passed to RTL, but is passed to a generator that generates RTL for this component.

Assigning Parameters

You can assign attributes to an object in any of the following ways:

- Using the Canvas View to Assign Parameters
- Using the Tabular View to Assign Parameters
- Using Tcl to Assign Parameters

Using the Canvas View to Assign Parameters

To assign a parameter to an object in the Canvas View, perform the following steps:

1. Click

2. in the GUI window to enter in the Canvas View.
3. Double-click on the object in the *Library Browser*.

   The object is loaded in the *Design Browser*.
4. Click the *Parameters* tab in the *Property Editor* pane.
5. Assign parameter details, such as name, type, and value as explained in the Using the Parameters Tab topic.

To specify more details, such as whether the parameter should be represented in the generated RTL, switch to the tabular view. See Using the Tabular View to Assign Parameters.

Using the Tabular View to Assign Parameters

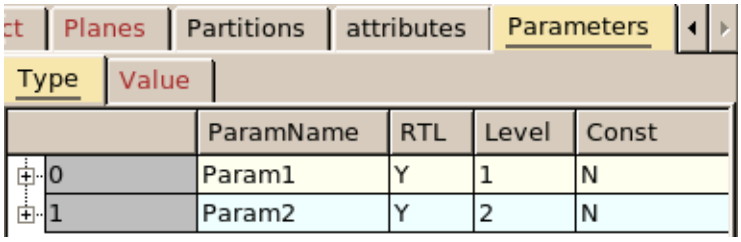
To assign parameter to an object in the Canvas View, perform the following steps:

1. Click

2. in the GUI window to enter in the tabular view.
3. Double-click on the object in the *Library Browser*.

   The object is loaded in the *Design Browser*.
4. Click the *Parameters* tab.
5. Right-click in the area under this tab, and click the *Insert Row* option from the shortcut menu.

   A new appears.
6. Click the *Type* tab.
7. Specify parameter details, such as name and type, as shown in the following figure:



In the above table, specify Y or N in the RTL column to specify if the parameter should be present in the generated netlist.

In addition, you can specify the level of parameters. For details, see Managing Dependant Parameters.

8. Click the *Value* tab.
9. Specify the default value of the added parameters in the table under the *Value* tab.

Using Tcl to Assign Parameters

Use the add_parameter Tcl command to assign parameters to objects.

Managing Dependant Parameters

Certain parameters may depend on the value of other parameters. To manage such dependant parameters, assign levels to the parameters.

The following table explains the dependency of different level parameters:

Level	Dependency
1	None
2	On Level 1 Parameters
3	On Level 2 or 1 Parameters
4	On Level 3, 2, or 1 Parameters

Consider an example in which the value of the P2 parameter is dependent on the value of the P1 parameter.

In this case, define the level of the P2 parameter as 2 and the level of the P1 parameter as 1.

You can specify the level of parameters in the tabular view. See Assigning Parameters.

Note:
For RTL flow, dependent parameters should not be used.

Working with 3-State Values

Note:
For RTL flow, do not use the 3-state values.

GenSys supports 3-state values to be used in common arithmetic/logical operations.

For example, a parameter can hold 3-state logic values, such as, 1x01x0. Parameters having 3-state values can be also be used in arithmetic/logical expressions. User generators can access fields having such 3-state values/expressions and GenSys returns the evaluated values of such expressions.

GenSys supports the following operations for 3-state values:

Operation	Operator used
Addition	+
Subtraction	-
Multiplication	*)
Division	/
Exponent	**
Modulus	%
Concatenation	concat
Bitwise negation	~
Bitwise and	&
Bitwise or	|
Bitwise xor	^
Bitwise equivalence	beq

Expressions involving 3-state values should be written in the following format.

```
=<operator> <value1> <value2>
```

The following examples show different operations on 3-state values:

- Parameter having 3-state value:

  ```
  add_parameter -name param1 -type LOGIC-value 1x11
  ```

- Addition operation:

  ```
  add_register -name reg1 -offset 0-width {=+ 1x10 1x11}
  ```
  Result: 10101

- Negation operation

  ```
  add_register -name reg1 -offset 0-width {=~ 101x}
  ```
  Result: 010x

- Concatenation operation

  ```
  add_register -name reg1 -offset 0-width {=concat 101x 1x11}
  ```
  Result: 101x1x11

- Expression having parameter reference

  ```
  add_register -name reg1 -offset 0-width {=+ 1x10 [p param1]}
  ```
  Result: 10101

4

Performing RTL Import

RTL import is a process in which RTL files (with precompiled libraries and packages) are read in GenSys and their corresponding IP-XACT component and design views are generated.

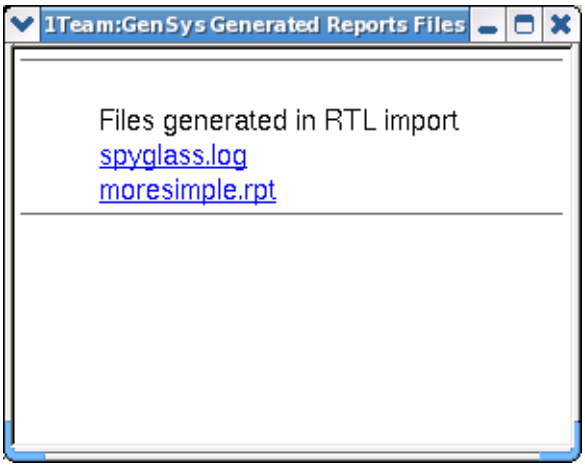
The IP-XACT component views primarily contain IP ports, IP parameters, and file-sets wrapped inside a synthesis model-view. The IP-XACT design views contain the same information as components along with the information of instantiated components and connectivity between instances and glue logic.

You can import RTL through GUI or Tcl. For details, see Importing RTL Through GUI and Importing RTL by Using Tcl Command.

When you import RTL through GUI:

- The status of RTL import appears under the *Log* tab of the Command/Log Pane.
- The details related to RTL import are saved in the GenSys_RTLlog directory.
- The information about any error or warning generated during RTL import is saved in log files.

  Links to these files appear in the *GenSys Generated Reports Files* window, as shown in the following figure:



You can click on the links to view the error or warning details.

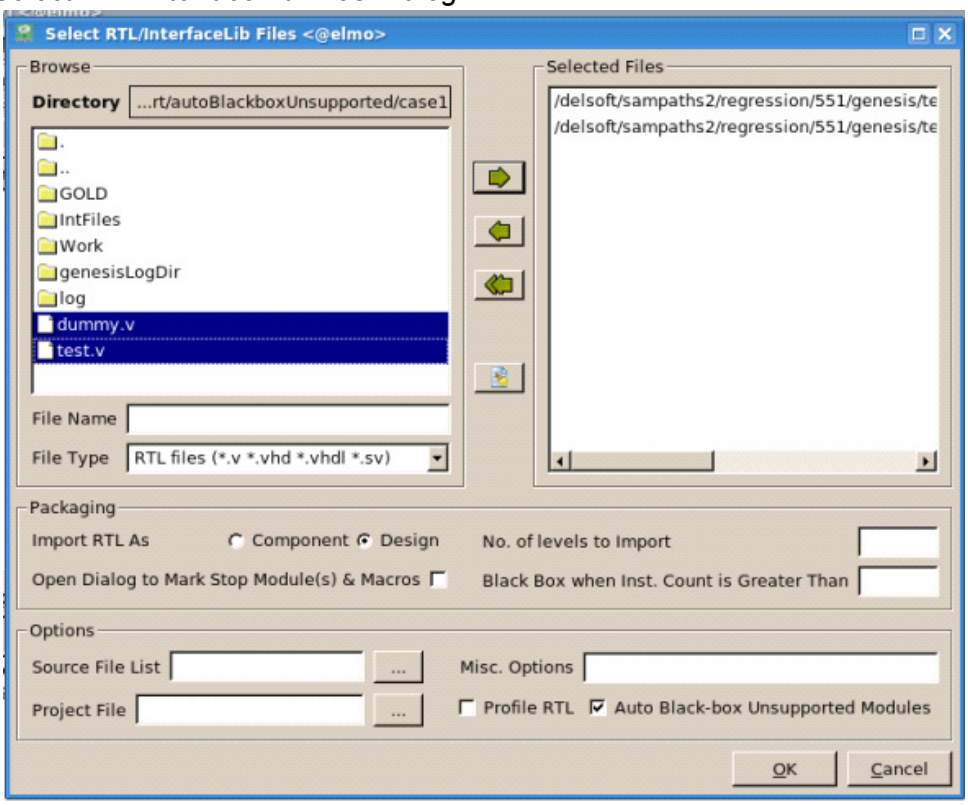
Importing RTL Through GUI

To import RTL through GUI, perform the following steps:

1. Select the *File* > *Load RTL* menu option.

   The *Select RTL/InterfaceLib Files* dialog appears, as shown in the following figure:

Figure 4-1 The Select RTL/InterfaceLib Files Dialog



2. Select the file type from the *File Type* drop-down list.
3. Select the files from the available list appearing in the dialog.
4. Click

5. to add the selected files.

   These files appear in the *Selected Files* section.

6. Select the *Component* or *Design* option from the *Import RTL As* section to specify that files being loaded are component or design RTL files, respectively.

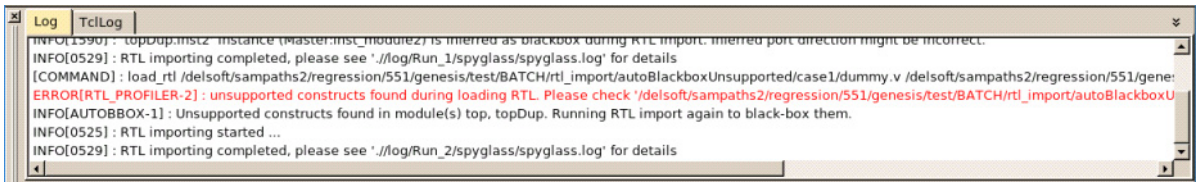
7. Specify an integer value in the *No. of levels to import* text box to specify the maximum level of hierarchies you want to import from the RTL.
8. Select the *Open Dialog to Mark Stop Module(s) & Macros* check box if you want to specify macros and configure stop modules.

   When you select this check box, the *Specify Stop Modules & Macros* dialog appears once you close the *Select RTL/InterfaceLib Files* dialog. For details, see Configuring Stop Modules and Macros.
9. Specify a black box count in the *Black Box when Inst. Count is Greater Than* text box.

   Modules having black boxes greater than the specified count are considered as black box.
10. Specify a source file in the *Source File List* text box.
11. Specify a project file in the *Project File* text box.

    For information on a project file, refer to the *Atrenta Console User Guide* of the SpyGlass help set.
12. Specify SpyGlass-specific options in the *Misc. Options* textbox. For example, while importing an RTL in VHDL, you can specify details, such as library-mappings using SpyGlass options.
13. Select the Auto Black-box Unsupported Modules check box to automatically black box modules with unsupported constructs. With this option enabled, RTL import is run with RTL Profiler to identify the unsupported constructs inside modules.

    The log window notifies you that RTL import will be run again to perform the actual black boxing identified in the first RTL import stage.

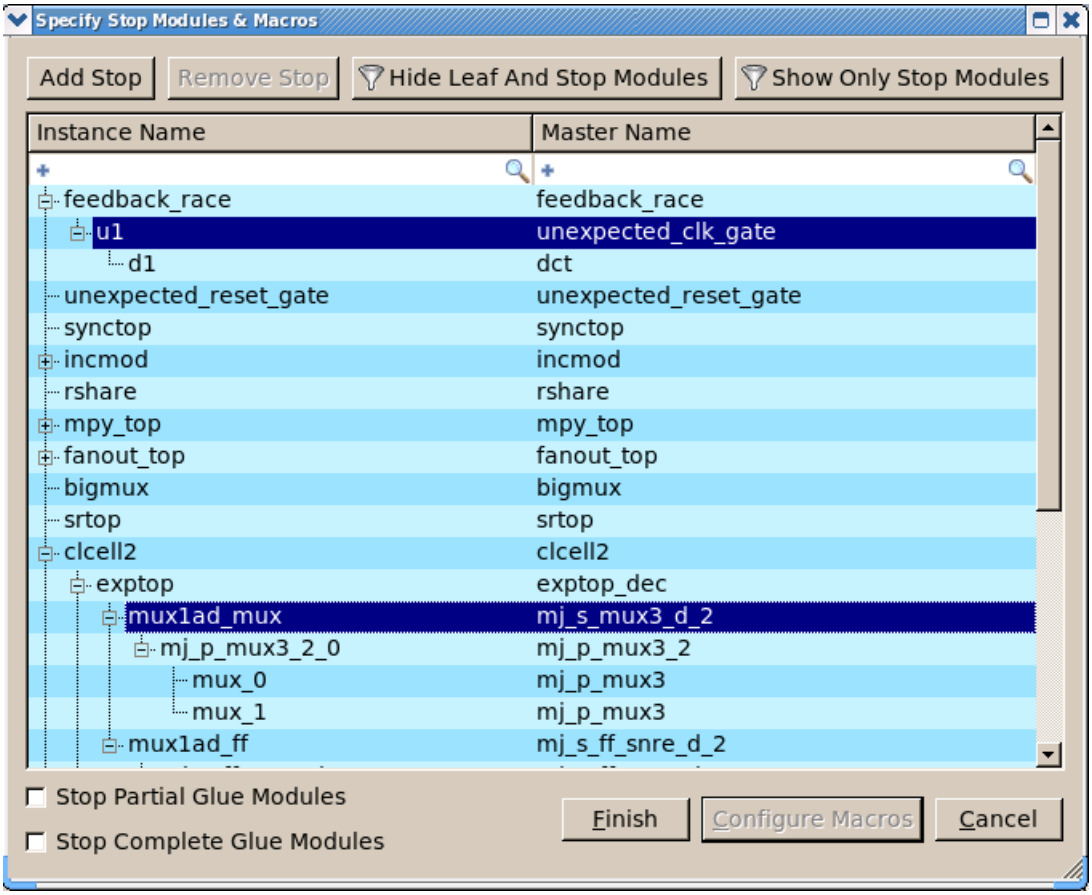


    In addition, a text file named blackboxed_modules.txt is generated in the log/Run_*/ directory with a list of modules black boxed.
14. Click the *OK* button to import the component/design RTL with the specified settings.

Configuring Stop Modules and Macros

The following figure shows the *Specify Stop Modules & Macros* dialog that appears once you click close *The Select RTL/InterfaceLib files Dialog*:

Figure 4-2 The Specify Stop Modules & Macros Dialog



Use the above dialog to configure Stop Modules and macros. For details, see:

- Configuring Sub-Systems as Stop Modules
- Configuring Macros

Configuring Sub-Systems as Stop Modules

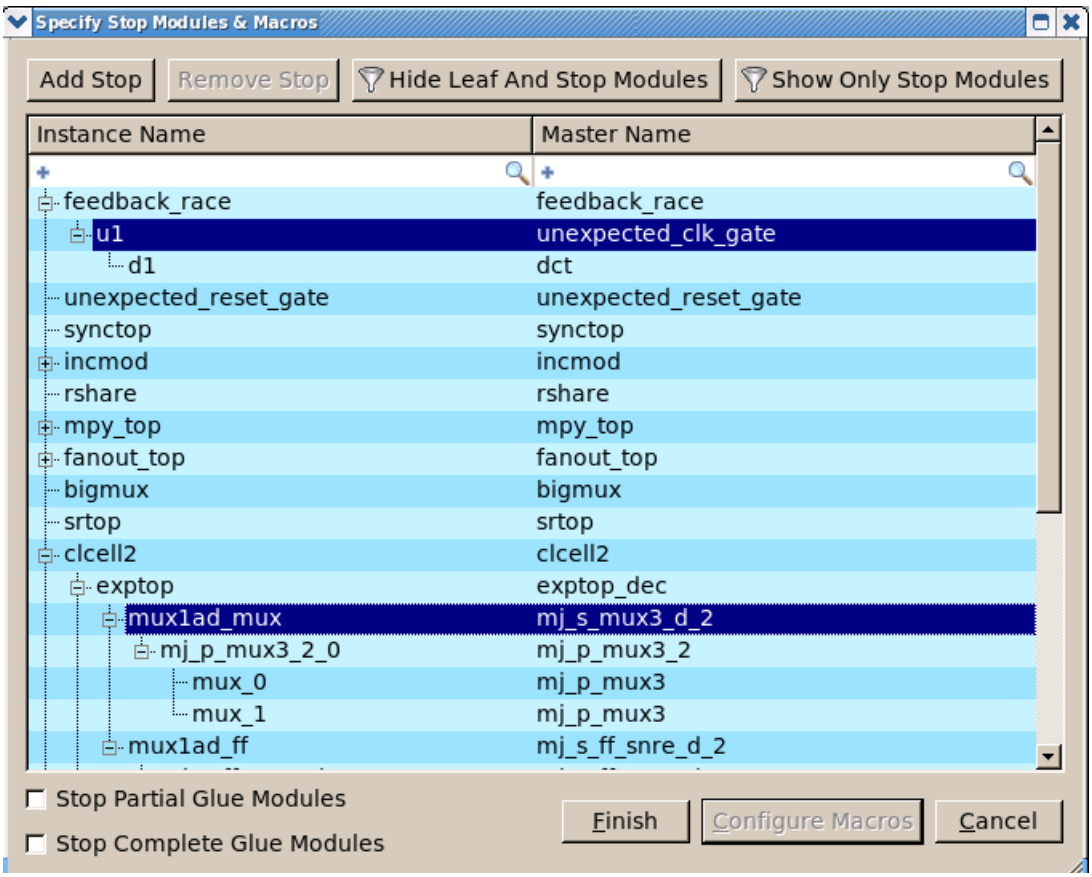
In the dialog shown in Figure 4-2, perform the following steps to specify sub-systems as Stop Modules:

1. Click the sub-systems keeping the <*Ctrl*> keyboard key pressed.
2. Click the *Add Stop* button.

Alternatively, right-click on the selection and select the *Add Stop* option from the shortcut menu.

3. Click the *Finish* button.

For example, consider the following figure showing selected sub-systems in the *Specify Stop Modules and Macros* dialog:



In the above dialog, when you click the *Add Stop* button:

- The selected sub-systems and the hierarchy below these sub-systems are not imported.
- These sub-systems appear as black boxes under the Design Browser Tab.

Specifying All Leaf-Level Modules as Stop Modules

To specify all the leaf-level modules as Stop Modules, right-click in the *Specify Stop Modules and Macros* dialog, and select the *Add Stop To All Leaf Modules* option from the shortcut menu.

Specifying Complete or Partial Glue Modules as Stop Modules

A complete glue module is a module that contains only glue logic inside it. However, a partial glue module contains at least one glue logic along with other instances.

To specify complete glue modules as Stop Modules, select the *Stop Complete Glue Module* check box in the *Specify Stop Modules and Macros* dialog.

To specify partial glue modules as Stop Modules, select the *Stop Partial Glue Module* check box in the *Specify Stop Modules and Macros* dialog.

If you select both the check boxes, GenSys considers only partial glue modules as Stop Modules.

Removing the Stop Specification

To remove the stop specification from sub-systems, perform the following steps:

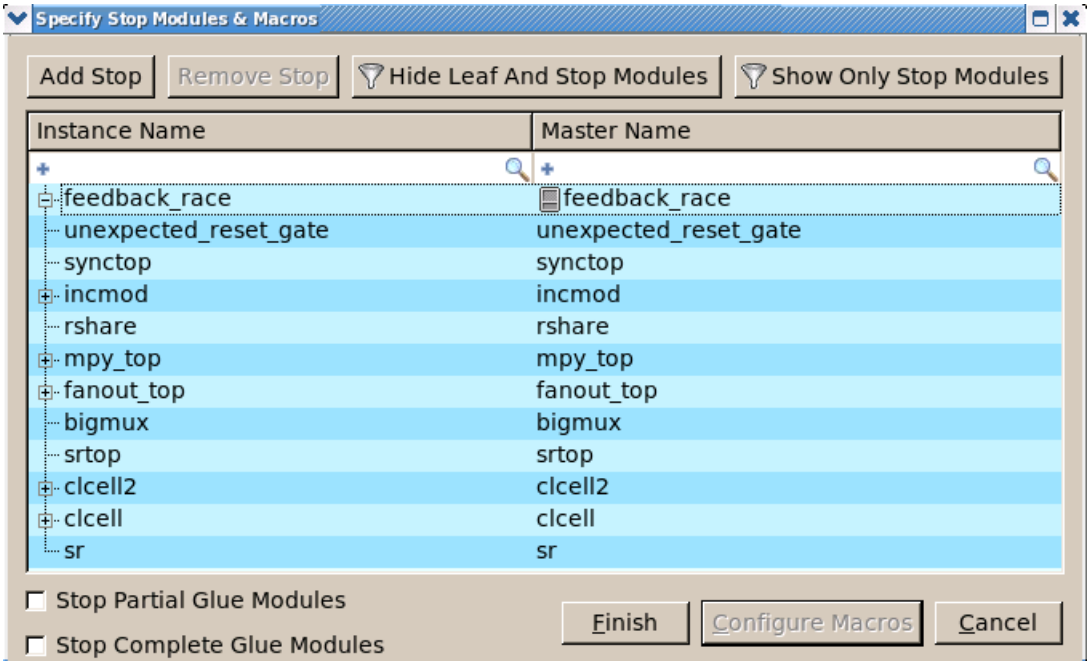
1. Click the sub-systems keeping the <*Ctrl*> keyboard key pressed.
2. Click the *Remove Stop* button.

To remove the stop specification from all the leaf-level modules, right-click in the *Specify Stop Modules and Macros* dialog, and select the *Remove Stop To All Leaf Modules* option from the shortcut menu.

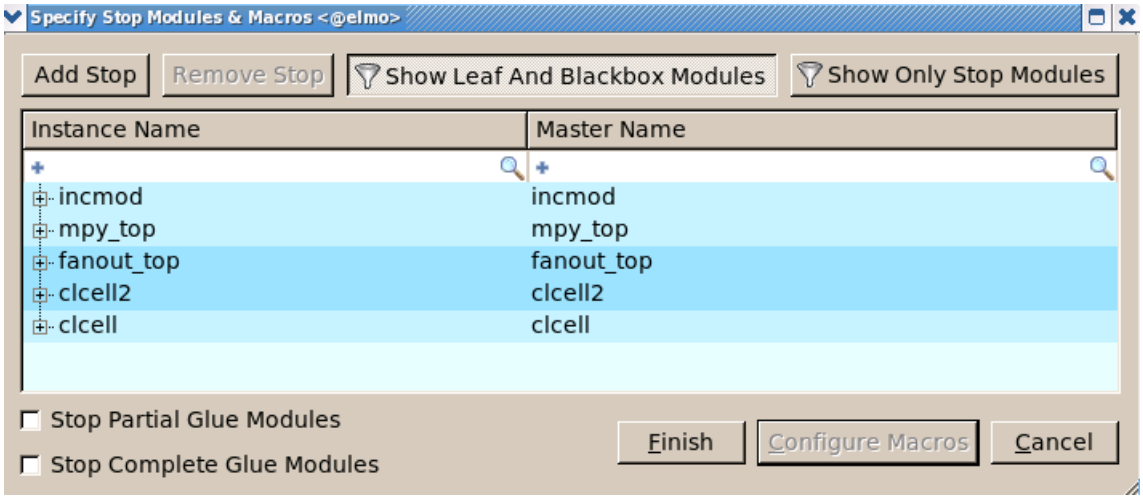
Hiding Leaf-Level and Stop Modules

To hide leaf-level and Stop Modules from the *Specify Stop Modules and Macro* dialog, click the *Hide Leaf and Stop Modules* button in this dialog.

For example, consider the following figure:



In the above dialog, when you click *Hide Leaf and Stop Modules* button, the leaf level and Stop Modules get hidden, as shown in the following figure:



In the above dialog, click the *Show Leaf and Blackbox Modules* button to show the hidden modules.

Viewing Only Stop Modules

To view only Stop Modules in the *Specify Stop Modules and Macros* dialog, click the *Show Only Stop Modules* button in this dialog.

Configuring Macros

For details, see Configuring Macros During RTL Import.

Importing RTL by Using Tcl Command

You can import RTL of a design or component by using the load_rtl Tcl command. For details on the load_rtl Tcl command, refer to *Tcl Command Reference Guide*.

Please note the following points:

- While importing components and design simultaneously, do not specify the -component argument of the load_rtl command.
- While importing RTL of a design, if you specify the -component argument, GenSys considers that design as component and no connectivity and instance information is populated down the hierarchy.
- If you do not specify any interface library file in the load_rtl command, GenSys uses interface libraries present in the path specified by the set_library_path Tcl command.

Merging Functionalities of Existing and Imported Object

Note:
This feature is not applicable to the RTL flow.

During SoC development, different designers work on different parts of the design. In many cases, they create a placeholders of some functionalities to be implemented later. For example, a designer may initially create an object with some ports and few interface-mappings and decide to implement the rest of the functionality later.

Now it may happen that while loading RTL of some objects, VLN of an object being imported matches with the VLN of an existing object for which the functionality is not completely defined. In such cases, you may want to perform RTL import to replace the existing object of the same VLN and also retain the default settings of the existing object in the new object. For example, you may want to retain the default port values and interface-mappings of the old object.

In such cases, set the -merge_views argument of the load_rtl Tcl command to TRUE. Setting this argument to TRUE forces GenSys to perform a clean import of the RTL hierarchy of the new object and also retain the properties of the object being replaced.

In this case, the VLNV of the new object will be same as the old object.

Performing Force RTL Import

During RTL import, GenSys compares the VLN (Vendor, Library, and Name) of an object being imported with the VLN of an object already present in the *Library Browser*. If this information matches for both the objects, GenSys does not performs the RTL import of that object and considers the already loaded object present in the *Library Browser*.

However, you can forcefully perform a clean import even if an object of the same VLN (Vendor, Library, and Name) is already present in the *Library Browser*. To do so, use the -import_all argument of the load_rtl Tcl command. Using this argument performs an RTL import of the complete object hierarchy.

If you use the -import_all argument, you must specify a Tcl function name in the -version argument of the load_rtl Tcl command. This Tcl function should return a version number of the new object being imported.

Given the VLN of the object being imported, all versions present for that VLN and version that is currently instantiated should be passed as arguments to this function.

Based on different situations, GenSys performs appropriate actions, as discussed below:

- If you do not specify the -version argument, GenSys invokes the default Tcl function, VLNV_next_version.
- If you specify a Tcl function name in the -version argument but do not define that Tcl function, GenSys does not perform the RTL import process for that object and reports an error.
- If the specified Tcl function returns the same version as that of the existing object, GenSys does not perform the RTL import of that object and considers the object already present in the *Library Browser*.
- If the specified Tcl function returns a new version, GenSys performs the complete import of the RTL hierarchy of the object, and assigns the new version to that object.

  Note:

  If you specify the -import_all and -merge_views TRUE arguments together in the load_rtl Tcl command, the -import_all argument is given priority.

Inferring Glue Logic During RTL Import

During RTL import, most of the data being imported is structural RTL. However, this data may also contain some behavioral logic, which is also referred to as glue logic.

Glue logic encountered while importing RTL designs are tagged in separate pseudo hierarchical boundaries. GenSys dissolves these pseudo hierarchical boundaries during RTL generation.

Importing Glue Logic

A glue logic refers to a collection of gates in the process/always blocks or in function/procedure calls. It may also be a set of signals of complex user-defined types.

By default, glue logic is imported during RTL import. While importing structural RTL during RTL import, if you do not want to import glue logic, set the -import_glue argument of the set_rtlimport_option Tcl command to FALSE.

GenSys then creates a separate glue component for all the glue logic present in a hierarchy. For example, during Verilog RTL import, GenSys creates a single glue component for all the always blocks and concurrent assign statements. Similarly, during VHDL import, GenSys creates a single glue component for all the process blocks and concurrent assign statements.

Creating Separate Glue Components

To create a separate glue component for each always/process block, use the -merge_glue FALSE option of the set_rtlimport_option TCL command. However, this option is added for a specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

Note:

This glue component is a pseudo hierarchy which is visible in schematic and the IPXACT/MRS/TCL database, while the glue component gets dissolved during the RTL generation.

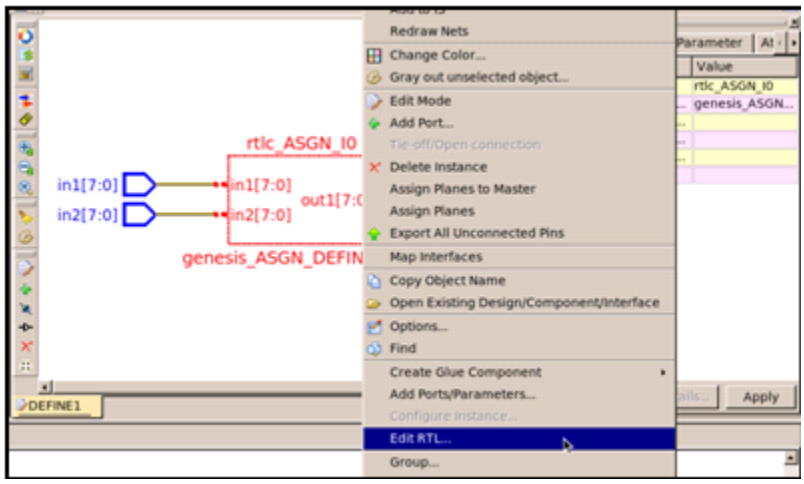
See Example of Inferring Glue Logic.

Note:

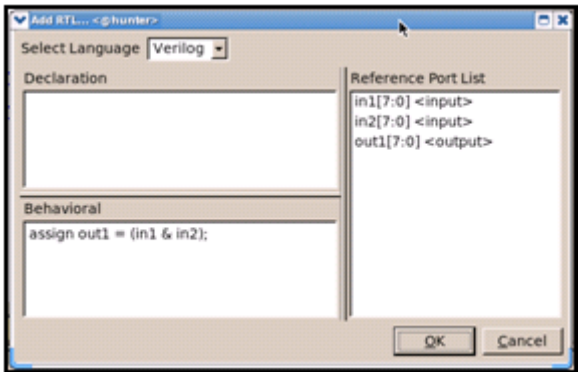
Normal buffer connections (simple assign statements) and tieoff connections do not get moved down into a separate glue module.

Editing Glue Logic

To edit the glue logic, right-click on the glue logic in the canvas view, and select the *Edit RTL* option from the shortcut menu, as shown in the following figure:



The following dialog appears in which you can edit the RTL of the selected glue logic:



Handling Complex Port Types

If a leaf-level VHDL glue component has complex port types, that is, user-defined types or any complex type other than std_logic and std_logic_vector, GenSys creates a VHDL wrapper module over that imported glue component.

Note:

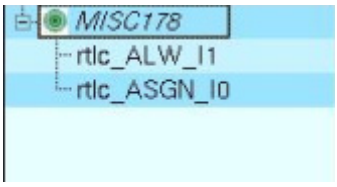
If a complex port type (other than std_logic and std_logic_vector) is present on a sub-system, other than a leaf-level component, GenSys flattens such ports during RTL import.

Example of Inferring Glue Logic

Consider a Verilog module, as shown in the following example:

```
module MISC178 ( out, out1, rst, en1, in1,in2,p);  parameter    P   =
20;  output [3:0] out;  output [3:0] out1;  input rst, en1;  input [3:0]
in1;  input [1:0] in2;  input   [P - 13:0]   p;  reg [3:0] out;  assign
out1 = in1 & in2;  always @(en1 or in1 or rst or p)    if (rst)
out[1:0] <= 2'b00;    else if (en1) out[1:0] <= in1;    else if (p)
out[1:0] <= in2;endmodule
```

The above module contains glue logic in the always and assign statements. Therefore, when run with the -merge_glue FALSE option, GenSys creates two separate glue components, rtlc_ALW_I1 and rtlc_ASGN_I0, as shown in the following figure:



Inferring Interfaces on Imported RTL

Use the infer_interfaces Tcl command to infer interfaces on the imported RTL.

This steps instantiates interfaces on the imported RTL with the properties specified by using the add_interface_mapping Tcl command. In addition, GenSys infers additional interfaces based on the pattern specified by the interface matching rules.

When more than one interface matches the actual ports of a component, the interface having minimum accumulative port width difference with respect to actual ports is instantiated.

Consider the following example:

```
interface : inf_64input [64:0]     L1;output [64:0]   L2;interface:
inf_32input [32:0]       L1;output [32:0]     L2;component: compinput
[7:0]      L1_1;output [7:0]    L2_1;
```

In the above example, the inf_32 interface has less accumulative port width difference than inf_64. Therefore, the inf_32 interface is instantiated on the component.

Checks Performed While Inferring Interfaces

To infer interfaces on a component, GenSys uses the instantiate_interface Perl routine. This Perl routine performs the following checks:

- For each interface of a component, logical port names are matched with the actual port names including the prefix and postfix.

  Note:
  The actual port name should be in the following format:
  <prefix><logical-port-name><postfix>

- In case of master/slave interface, in addition to the required ports (R), the required in master (RM)/ required in slave (RS) logical ports should match to the corresponding actual ports of component. This port matching is strict and in case interface contains any required type (R/RO/RS) port and it is not mapped to an actual port, that interface is not instantiated.
- Port matching for all optional ports of interface is done. But, this is not a strict checking and optional ports can be left unmapped.
- LSB and MSB of actual ports of component should be within the range of LSB and MSB of logical ports of interface. This is because actual ports can have width less than the width of logical ports of interface.
- An interface that has maximum number of match is inferred on the component.
- Name of the interface class in component will be in the *<prefix><interface-lib-name><postfix>* format.

  Consider the following example:

  ```
  interface: intfinput [0:7] L1;input [0:7] L2;input [0:7] L3;component:
  compinput [0:7] L1_1;input L2_1;input [0:3] L3_1;
  ```

  In the above example, the intf interface would be inferred in the component, comp. The name of the inferred interface would be intf_1.

Inferring IP-XACT Interfaces

GenSys also infers IP-XACT interfaces. An IP-XACT interface is matched in MASTER, SLAVE, SYSTEM and their mirrored versions.

The priority of inferring an interface on a component is as follows:

1. MASTER
2. SLAVE
3. MASTER mirrored

4. SLAVE mirrored
5. SYSTEM
6. SYSTEM mirrored

Inferring Interface Connections

You can infer interface connections on a design by using the infer_interface_connections Tcl command.

Preserving Net Names after Updating the Imported RTL

When you generate a netlist after updating an imported RTL, GenSys can preserve net names of the original RTL.

Preserving net names in the generated netlist is important because the original RTL may have associated constraints in separate files. If net names are not preserved, you may have to re-write these constraints based on new net names present in the generated netlist.

Note:
Net names are preserved only for complete connections. For example, if certain bits, say n1[1:o], of the vector net n1[3:0] connect a source and destination, n1[1:o] is not preserved in the generated netlist. However, if the complete vector net n1[3:0] connects the source and destination, net n1[3:0] is preserved.

Note:
For VHDL RTL, net name is not preserved when any of the source/destination has a complex type port, that is, other than std_logic.

Examples

Example 1

Consider the following RTL imported in GenSys:

```
module leaf1(A);  output [1:0]A;endmodule
module leaf2(B,C,D);  input [1:0]B;  input [1:0]C;  input
[1:0]D;endmodule
module top;  wire [1:0]n1;  wire [1:0]n2;  leaf1 I1(n1);  leaf2
I2(n1,n2,n2);  assign n2 = n1;endmodule
```

The following netlist is generated for the above RTL:

```
module top ();wire   [1:0]  n1;wire   [1:0]  n2;   leaf1   I1 (
.A(n1)     //O:2                );    leaf2    I2 (
.B(n1)  ,  //I:2                  .C(n2)  ,  //I:2
.D(n2)     //I:2                );    assign n2 = n1;endmodule
```

In the above netlist, original net names are preserved based on the imported RTL.

Example 2 - Preserving Net Names when Sink Side of a Connection is Partially Connected

For a connection in an RTL, driver side should be completely connected with a net. However, the sink side can be partially connected.

Consider I1.a is the driver and I2.b is the sink. Therefore, net name are preserved in the following cases:

- Case 1

  `comp1 I1(.a(n1));comp2 I2(.b(n1));`
  In this case, net name n1 is preserved and generated RTL will also look like it.

- Case 2

  `comp1 I1(.a(n1));comp2 I2(.b{n1,2'b11});`
  In this case, net name n1 is preserved since this is completely connected to driver side (I1) and on sink you have fanins other than n1.

- Case 3

  `comp1 I1(.a(n1));comp2 I2(.b{n1(1:0),2'b11});`
  In this case, net name n1 is not preserved. It is completely connected to source however not all bits of n1 are going to sink.

- Case 4

  `comp1 I1 (.a(n1));comp2 I2(.b(n2));comp3 I3(.c{n1,n2});`
  Consider I1.a and I2.b are drivers. Then, both n1 and n2 net names are preserved in this case.

Example 3 - Preserving Net Names in Case of Tieoff Connectivity

For tieoff connectivity in an input RTL, net names are preserved only when a complete tieoff net is driving a sink. This is explained in the following example:

```
wire [10:0] n1 = 10'b1111100001;comp1 I1 (.a(n1)); //n1 is preservedcomp2
I2(.b{n1,4'b1011}); // n1 is preservedcomp3 I3 (.c(n1[2:0])); //n1 is not
preserved.
```

Handling Verilog Pre Processor Directives

RTL designers use Verilog pre processor directives, such as '`ifdef, `ifndef, `else, `elsif, and `endif to express RTL intent.

During RTL import, GenSys imports the objects under these directives and preserves appropriate macro combinations applied on them.

During Verilog RTL generation, all such objects are placed under the same macro combination in which they were present in the original Verilog RTL.

For example, the following Verilog file uses the 'ifdef directives to hide or show certain ports:

Figure 4-3 Sample RTL File Containing Macro Configurations

```
module test #(parameter p1=2,p2=4, p3=8)(
  `ifdef TWO_BIT
    input [p1-1:0] I2,
    output [p1-1:0] O2
  `elsif FOUR_BIT
    input [p2-1:0] I4,
    output [p2-1:0] O4
  `elsif EIGHT_BIT
    input [p3-1:0] I7,
    output [p3-1:0] O7
  `else
    input I1,
    output O1
  `endif
  );
endmodule
```

Based on the configurations specified by the directives in the above RTL, GenSys provides you the flexibility of:

- Loading Configurable Objects
- Enabling and Disabling Macros During RTL Import
- Creating Configurable Objects

Loading Configurable Objects

To load objects that are under some configuration, perform the following steps:

1. Specify the following Tcl command:

   ```
   set_rtlimport_option -preserve_configurations TRUE
   ```

2. Load the RTL.

After performing the above steps, the applicable macro configurations are preserved in all the loaded objects.

Note:
This flow is memory and runtime intensive because it brings all the objects in GenSys, which are by default not visible and are under some conditional Verilog pre processor directives. To achieve this, the design read step is called multiple times to make all objects visible at least once.

You can selectively bring objects of specific configurations by enabling/disabling some macros during RTL import through GUI. For details, see Enabling and Disabling Macros During RTL Import.

Enabling and Disabling Macros During RTL Import

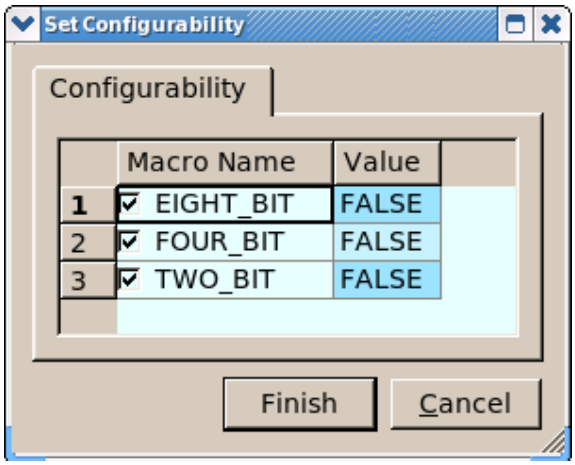
In GUI, you can selectively enable or disable global macros that impact configurability values of objects in the input RTL.

To do so, perform the following steps:

1. Set the -preserve_configurations argument of the set_rtlimport_option Tcl command to TRUE.
2. Load the RTL (such as in Figure 4-3) by performing any of the following actions:
   - Select the *File -> Load RTL* menu option.
   - Specify the -ui argument with the load_rtl Tcl command.

   On performing any of the above actions, the *Specify Stop Modules & Macros* dialog appears.
3. Click the *Configure Macros* button in this dialog.

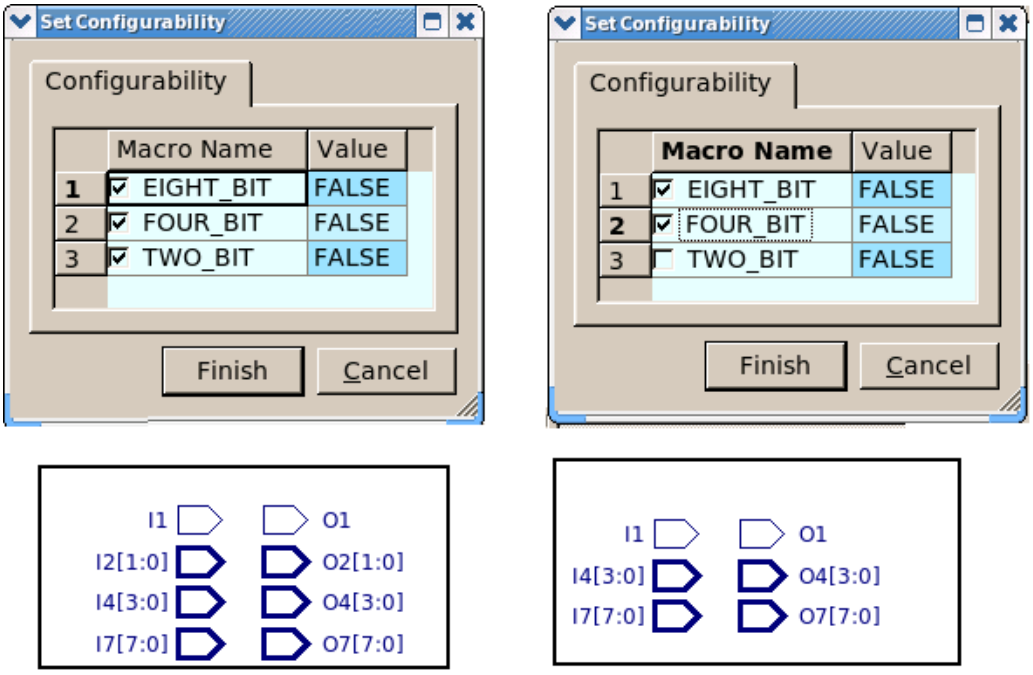
   The *Set Configurability* dialog appears, as shown below:



The above dialog shows the global macros that impact the `ifdef configurability values of the objects defined in the input RTL.

4. Select or deselect macros in the above dialog to merge or ignore objects under the selected macro configurations.
5. Click the *Finish* button.

   After performing the above steps, objects under the selected configuration settings are loaded in GenSys while objects under the deselected macros are ignored and not loaded. For details, see the following figure:



Creating Configurable Objects

You can specify a configurability for objects while creating them in Tcl such that based on the configurability, GenSys places those objects under appropriate Verilog conditional pre-processor directives (such as `ifdef) while generating verilog RTL.

For example, consider the following commands:

```
new component master1
add_port -name a -direction IN -align MSB -lsb_bit RIGHT-macro_visibility
{X:A}
add_port -name b -direction OUT -align MSB -lsb_bit
RIGHT-macro_visibility {X:A}
```

The above commands create the ports a and b and define a configurability for them by using the -macro_visibility argument. This configurability indicates that when both the X and A macros are TRUE, the ports a and b should be visible.

Based on this configurability, ports a and b are placed under appropriate `ifdef statements in the generated Verilog RTL, as shown below:

```
module master1 (              `ifdef X                `ifdef A
a,                    b,                  `endif
`endif);`ifdef X    `ifdef A    input          a;     output         b;
`endif`endifendmodule
```

You can specify the -macro_visibility argument for the following objects:

Object Type	Command to Create this Object
Ports	add_port
Parameters	add_parameter
Connection	add_connection
Instances	add_instance
Parameter override	set_parameter
Include files	add_file

Reporting Unsupported Constraints in GenSys

GenSys is an RTL modification and creation engine in which you read the RTL (load_rtl), perform RTL modifications, and then generate the modified RTL.

However, the modified RTL may not be correctly generated due to the presence of some unsupported constructs in the original RTL.

Therefore, the need is to report such constructs so that the user performs the required actions, such as remove such constructs or stop the modules containing such constructs.

This section describes the constructs unsupported in GenSys and the RTL profiler report that shows information about these constructs. For details, see Unsupported Constructs in GenSys and The RTL Profiler Report.

Unsupported Constructs in GenSys

This section describes the unsupported constructs in the VHDL, Verilog, and SystemVerilog flow. For details, see:

- Unsupported Constructs in the VHDL Flow
- Unsupported Constructs in the Verilog Flow
- Unsupported Constructs in the SystemVerilog Flow

Unsupported Constructs in the VHDL Flow

The following VHDL constructs used inside process blocks are supported, but unsupported if used outside the process blocks:

- Linkage ports in VHDL
- Usage of VHDL alias constructs
- Usage of VHDL extended identifier
- Usage of VHDL attributes
- Usage of unsynthesizable constructs based on keywords only
- Usage of VHDL configuration and if it has a specific binding, the usage becomes illegal
- Tieoff connections to non std-logic types
- Type casting across connections except for signed, unsigned, and std_logic_vector
- Generics with parameterized user defined types

Unsupported Constructs in the Verilog Flow

The following Verilog constructs are not supported in GenSys:

- Presence of `ifdef on `define constructs if the -preserve_configurations argument of set_rtlimport_option is set to TRUE.

  `ifdef on `define constructs are shown as unsupported in The RTL Profiler Report if it is present inside a module. Else, it will be shown as an audit information in this report. (See Example of ifdef on `define, `undef, `timescale)

- Presence of `ifdef on `undef constructs if the -preserve_configurations argument of set_rtlimport_option is set to TRUE. (See Example of ifdef on `define, `undef, `timescale)

  `ifdef on `undef constructs are shown as unsupported in The RTL Profiler Report if it is present inside a module. Else, it will be shown as an audit information in this report.

- Presence of `ifdef on `timescale, timeunit, and timeprecision constructs if the -preserve_configurations argument of set_rtlimport_option is set to TRUE. (See Example of ifdef on `define, `undef, `timescale)

  It is reported as unsupported in The RTL Profiler Report if it is present inside a module. Else, it will be shown as an audit information in this report.

- Presence of `ifdef on a complete module if the -preserve_configurations argument of set_rtlimport_option is set to TRUE (See Example of `ifdef on Complete Module)

- Presence of `ifdef on assertions, initial blocks, and unsynthesizable constructs if the -preserve_configurations argument of set_rtlimport_option is set to TRUE (See Example of `ifdef on Assertions, Initial Blocks, and Unsynthesizable Constructs)
- Presence of `ifdef on partial expressions/statements if the -preserve_configurations argument of set_rtlimport_option is set to TRUE.

  (See Examples of `ifdef on Partial Expressions/Statements)
- Presence of local `undef constructs (See Example of Local `undef Constructs)
- Presence of multiple definitions of the same macro (See Example of Multiple Definitions of the Same Macro)
- Presence of multiple `ifdef branches in the same line if the -preserve_configurations argument of set_rtlimport_option is set to TRUE.

  This is shown in Example 2. Also see Example 1.
- Presence of pragmas in the same line as RTL constructs

  This is shown in the following example:

  ```
  input a; /* synopsys translate_off */ initial begin a = 1; end /*
  synopsys translate_on */
  ```
  (Also see Example of Pragmas in the Same Line as RTL Constructs)
- Presence of RTL code for unsynthesizable constructs in same line as the macros.

  (See Example of RTL Code for Unsynthesizable Constructs in Same Line as Macros)
- Presence of the defparam RTL construct (See Example of defparam/Usage of real Keyword in AMS)
- Presence of gate primitives, such as buf and bufif1 (See Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations)
- Presence of AMS constructs, such as the real keyword (See Example of defparam/Usage of real Keyword in AMS)
- Presence of escaped identifiers (See Example of Escaped Identifiers)
- Presence of supply0 and supply1 net types (See Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations)
- Presence of ports with implicit net declarations, as shown in the following example:

  ```
  module abc (in1 [1:0], out1);
  ```
  (Also see Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations)

Unsupported Constructs in the SystemVerilog Flow

The following SystemVerilog constructs are not supported in GenSys:

- Presence of `ifdef on SystemVerilog interfaces if the -preserve_configurations argument of set_rtlimport_option is set to TRUE.
- Presence of unsynthesizable SystemVerilog constructs
- Presence of the extern keyword in module definition
- Presence of nested modules and nested interfaces
- Presence of glue logic in interface definitions
- Presence of a parameter/macro of the same name inside the module definition as in the global scope

---

The RTL Profiler Report

The RTL profiler is a text report that shows the following information:

- The RTL constructs unsupported in GenSys
- Impact of GenSys switches on RTL constructs
- Information on various constructs present in the RTL

Generating the RTL Profiler Report

To generate this report, perform the following actions:

- Specify the set_rtlprofiler_options Tcl command as shown below:

  ```
  set_rtlprofiler_options  -file <report-name>  -hierarchy <TRUE |
  FALSE>  -report < all | unsupported | constructs>  -detail <TRUE |
  FALSE>
  ```

  The details of the above arguments are as follows:

  - -file *<report-name>*: Specifies the name of the RTL profiler report.
  - -hierarchy *<TRUE | FALSE>*: Specifies whether the report is generated for the top-level design only or should be run hierarchically on all the modules.
  - -report *<all | unsupported | constructs>*: Specify all to report all the design properties.

  Specify unsupported to report all the unsupported GenSys constructs. For information on these constructs, see Unsupported Constructs in GenSys.

  Specify constructs to report the constructs that can be useful in understanding the RTL and for which GenSys can suggest the recommended switches to be used. For

information on these constructs, see Information on various constructs present in the RTL.

  - -detail <*TRUE* | *FALSE*>: Set this argument to FALSE to report a concise view of information. To view the detailed information for better debugging, set this argument to TRUE.

- Set the -profile_rtl argument of the load_rtl Tcl command to TRUE to generate the report during RTL load.

The RTL constructs unsupported in GenSys

To know these constructs, see Unsupported Constructs in GenSys.

Impact of GenSys switches on RTL constructs

This section of the report provides the following information:

- Presence of `ifdef constructs in the RTL when the -preserve_configurations argument of set_rtlimport_option Tcl command is not set to TRUE.
- Presence of unsynthesizable constructs -preserve_unsynth_constructs argument of set_gensys_options Tcl command is not set to TRUE.
- Presence of pragmas and when the -preserve_pragmas argument of set_rtlimport_option Tcl command is not set to TRUE.
- Presence of glue logic and the usage of the -stop_on_partial_glue_modules or -stop_on_complete_glue_modules arguments of the set_rtlimport_option Tcl command.
- Presence of instances in a module when the number of modules is greater than what is specified in the -instance_threshold <*num*> argument of the load_rtl Tcl command or the default argument value.

Information on various constructs present in the RTL

This section of the report provides the information that helps the user to understand the RTL better:

- Presence of `ifdef: The report captures various `ifdef macros used, various `ifdef macro combinations, and the `ifdef macro combination per module. (See Example on the Usage of ‘ifdef and ‘define)
- Usage of `define: The report captures the location where the `define pre-processor directives are defined (file name), total count per file, and the total count per module. (See Example on the Usage of ‘ifdef and ‘define)

- Presence of parameters and generics: The report captures the total number of parameters/generics per module. (See Example on Generics and Local/Complex/String Parameters)
- Presence of string parameters (See Example on Generics and Local/Complex/String Parameters)
- Presence of parameters of complex types (anything that is not simple integer type) (See Example on Generics and Local/Complex/String Parameters)
- Number of local parameters (See Example on Generics and Local/Complex/String Parameters)
- Presence of parameterized multi-dimensional ports and wires (See Example of Parameterized Multi-Dimensional Ports/Wires)
- Presence of complex user-defined ports (See Example of Complex User-Defined Ports)
- Presence of glue logic in a module (See Example on the Presence of Generate Blocks)
- Count of generate blocks in a module (See Example on the Presence of Generate Blocks)
- Count of generate-if, for, and case constructs in a module (See Example on the Presence of Generate Blocks)
- Presence of array-of-instances (See Example on the Presence of Array-of-Instances)
- Partial assignments inside glue logic or generate logic (See Example of Partial Assignments in Glue/Generate Logic)
- Presence of VHDL modules with complex types instantiated inside Verilog or SystemVerilog modules (See Example Of VHDL Modules of Complex Types)
- Presence of `define macros in any place except port msb/lsb, parameter value, parameter map, port map, glue logic, and instance master name
- Presence of local `define constructs (See Example of Local ‘define Constructs)

Examples of Constructs to Help Better Understand the RTL

This section covers the following examples:

- Example on the Usage of ‘ifdef and ‘define
- Example on Generics and Local/Complex/String Parameters
- Example of Parameterized Multi-Dimensional Ports/Wires
- Example of Complex User-Defined Ports

- Example on the Presence of Generate Blocks
- Example on the Presence of Array-of-Instances
- Example of Partial Assignments in Glue/Generate Logic
- Example Of VHDL Modules of Complex Types
- Example of Local ‘define Constructs

Example on the Usage of ‘ifdef and ‘define

```
`ifdef x`define y 10`else`define Y 20`endif module test();parameter P1 =
`ifdef Z 10 `else 20 `endif ;`ifndef Wparameter P2 =10;`endif`ifdef
MM`ifdef NNparameter R = 10;`endif`endif`define HH 10`define GG
20endmodule
```

Example on Generics and Local/Complex/String Parameters

```
`define Y 2`include "inc.v"module test();`define W 20`include
"inc2.v"`ifndef Xparameter A1 = 2;`endifparameter A2 = 4'b1111;parameter
A3 = "abcsde";parameter A5 = 3'b000;localparam L1 = 10;endmodule
```

Example of Parameterized Multi-Dimensional Ports/Wires

```
`define N_PORTS 3`define N_CHANNELS 4 module pwr_ctrl(            input
logic                clk, rst_n,                 input   logic
[`N_PORTS-1 : 0] [2 : 0]  channel_addr_i,        input   logic
[`N_CHANNELS-1  : 0]          transaction_complete_i,        output
logic [`N_CHANNELS-1  : 0]       sleep,        output  logic [3:0][2:0]
run    );pwr_ctrl2 test0();endmodulemodule pwr_ctrl2(           input
logic                clk, rst_n,                 input   logic [`N_PORTS-1
: 0] [2 : 0]             channel_addr_i_2,           input   logic
[`N_CHANNELS-1  : 0][`N_PORTS-1 :           0]
transaction_complete_i_2,         output  logic [`N_CHANNELS-1  : 0]
sleep,           output  logic [3:0][2:0] run    );endmodule
```

Example of Complex User-Defined Ports

```
typedef struct {  logic [7:0] x;  logic [7:0] y;} s_point; // named
structuremodule m1 (output s_point outPoint);//endmodulemodule m2 (input
s_point inPoint);//endmodulemodule top ();    s_point point;    m1
m1_inst (point);    m2 m2_inst (point);endmodule
```

Example on the Presence of Generate Blocks

```
module top (clk, out);  input         clk;  reg     [3:0] count = 0;
wire    [8:0] path;  output        out;  always @ (posedge clk)    count
<= count + 1;  assign path[0] = count[3];  assign out = path[8];  genvar
n;  generate    for (n=0; n<=7; n=n+1) begin : flops      dflop u
(.clk(clk), .in(path[n]), .out(path[n+1]));    end
endgenerateendmodulemodule dflop (clk, in, out);  input         clk, in;
output reg    out;  always @ (posedge clk)    out <= in;endmodule
```

Example on the Presence of Array-of-Instances

```
module my_module (a, b, c);input a, b;output c;  assign c = a & b
;endmodulemodule top (a, b, c) ;input [3:0] a, b;output [3:0] c;
my_module inst [3:0] (a, b, c);endmodule
```

Example of Partial Assignments in Glue/Generate Logic

```
module test(output p);  reg [4:0] p;  always@(*)  begin    p[2:0] =
2'b100;  endendmodule
```

Example Of VHDL Modules of Complex Types

```
// top.v
module top();  InstExecMon InstExecMon();endmodule



//test.vhd
library ieee;use ieee.std_logic_1164.all;use ieee.numeric_std.all;use
ieee.std_logic_textio.all;library work;use work.pkg.all;entity
InstExecMon is    generic(        gc_PrintAddr : boolean := true
);    port(       nRst            : in std_logic;        Clk
: in std_logic;        DnovaCoreNr      : in std_ulogic_vector(7 downto
0);        Core2InstExecMon : in t_Core2InstExecMon        );end
InstExecMon;
```

Example of Local ‘define Constructs

```
`ifdef x`define y 10`else`define Y 20`endifmodule test();parameter P1 =
`ifdef Z 10 `else 20 `endif ;`ifndef Wparameter P2 =10;`endif`ifdef
MM`ifdef NNparameter R = 10;`endif`endif`define HH 10`define GG
20endmodule
```

Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations

```
module gates();  wire out0;  wire out1;  wire out2;  reg
in1,in2,in3,in4;  not U1(out0,in1);  and U2(out1,in1,in2,in3,in4);  xor
U3(out2,in1,in2,in3);endmodule
```

Example of Escaped Identifiers

```
// There must be white space after the// string which uses escape
charactermodule \1dff (q,      // Q output\q~ ,   // Q_out outputd,
// D inputcl$k,   // CLOCK input\reset* // Reset input);input d, cl$k,
\reset* ;output q, \q~ ;  endmodule
```

Example of Pragmas in the Same Line as RTL Constructs

```
/*synopsys synthesis_off*/ `define  x 10 /*synopsys synthesis_on*/
module test(
/*pragma translate_off*/ input p1 /*pragma translate_on*/
```

```
);
//synopsys translate_off
parameter P = 10; //synopsys translate_on
endmodule
```

Example of defparam/Usage of real Keyword in AMS

```
module my_module (Clk, D, Q) ;  parameter width = 2,  delay = 10 ;  input
[width - 1 : 0] D ;  input Clk ;  output [width : 0] Q ;  assign #delay Q
= D;endmodulemodule top;  reg Clk ;  reg [7:0] D ;  wire [7:0] Q
;  my_module inst_1(Clk, D, Q) ;endmodulemodule override ;  defparam
top.inst_1.width = 7 ;endmodule
```

Example of Multiple Definitions of the Same Macro

```
`define NN 1`define Y 20module test();`define RR 3`define NN 2`define X
10`define RR 4endmodule`define Z 30
```

Example of Local `undef Constructs

```
`undef Xmodule A();  `undef Yendmodule
```

Examples of `ifdef on Partial Expressions/Statements

Example 1

```
genvar i;`ifdef FPGA   for (i = 0; i < SIZE1; i = i +1)`else  for (i = 0;
i < SIZE2; i = i + 1)`endifbegin : sreggen   sreg # (.P(x)) master
(.clk(clk), .data (data))endendgenerate
```

Example 2

```
`ifdef BEHAVIOR
  always @(posedge clk1)
`else
 always @(posedge clk2)
 `endif
begin
out <= 8'b0;
end
```

Example 3

```
module test(I1,I2,I3,O1);input I1;input I2;input I3;output O1;assign O1 =
I1 & `ifdef X                    I2                `else
I3                 `endif                    ;endmodule
```

Example of RTL Code for Unsynthesizable Constructs in Same Line as Macros

Example 1

```
`ifdef XX initial begina = 1;end `endif
```

Example 2

```
always @(posedge clock)begin   `ifdef ALTERA_XCVR_ASSERTIONS assert
property   ($rose(reset_all) |=> $rose(pll_powerdown));`endif    end
```

Presence of Multiple `ifdef Branches in the Same Line

Example 1

```
`ifdef Y`define Z `ifdef R 10 `else 20 `endif`endifmodule
test();parameter P = `ifdef X 10 `else 20 `endif ;endmodule
```

Example 2

The following is the example of multiple ‘ifdef branches in the same line when the -preserve_configurations argument of set_rtlimport_option is set to TRUE.

```
assign a = `ifdef XX b&c `else d&e `endif;wire w1, `ifdef XX w2 `endif,
`ifdef YY w3 `endif;
```

Example of `ifdef on Complete Module

```
`ifdef Xmodule A ();`elsemodule A(input a1);`endif endmodule`ifdef
Ymodule B();endmodule`endifmodule top();`ifdef XA A0();`elseA
A0(1'b1);`endif`ifdef YB B0();`endifendmodule
```

Example of ifdef on `define, `undef, `timescale

```
module test();`ifdef N`timescale 10ns/1ns`else`timescale 100s/
1ns`endif`ifdef X`ifdef Y`define Z 10`endif`endif`ifdef XX`undef
Z`endifendmodule
```

Example of `ifdef on Assertions, Initial Blocks, and Unsynthesizable Constructs

```
`ifdef simtypeBehavEmureg [1023:0] mem_file_0;   initial begin
mem_file_0 = "lib/new_ram/ram_code/B13RF16X74RR1_0.txt";    #50;
$display("%0t: ** Info: Pre-Loading the RAM    B13RF16X74RR1_0.txt %0s",
$time, mem_file_0);     //$readmemb(mem_file_0, memory_0.mem);
End`endif
```

5

Managing Connections

Every user has its own preference for making design connections. However, it is recommended to follow the following approach to create connections:

1. Instantiate the components in the design with the appropriate parameters.
2. Export the pins of the instantiated components as top level ports.
3. Add additional top level ports.
4. Make interface connections.
5. Create the necessary ad-hoc connections.
6. Tie off any signals.
7. Check design connectivity.

Understanding Connection Types in GenSys

You can create the following types of connections in GenSys:

- Adhoc Connections
- Ad-hoc Tieoffs
- Interface Connections
- Unconnected
- Exported Interface
- Default Open
- Default Tieoff

Adhoc Connections

Adhoc connections are also known as Pin to Pin connections. In Adhoc connections, each port of an instance, for example I1, is directly connected to the ports of another instance, I2.

You need to specify the source pin, destination pin, source LSB, and destination MSB.

When connections are specified to/from a component instance for which -allow_create option is set to Y and the required ports and interfaces do not exist, they are created on the component. The direction of these auto-created ports will be inferred from the other side of the connection. These ports can be scalar or vector according to the requirement of the specified connections.

If user has already specified a port on the component with -allow_create option set to Y, the LSB/MSB of that port will be modified according to the connectivities specified at the design level for that port.

Ad-hoc Tieoffs

Ad-hoc tie-off connections are those connections in which the ports of an instance are tied to a specific value, such as 0, 1 etc. instead of a pin.

To configure the ad-hoc tie-off connections for an instance, right-click on an instance under the Design Browser, and select the Reconfigure Tie off connection option from the menu. This displays the Configure TieOff Values dialog. See figure Figure 5-1.

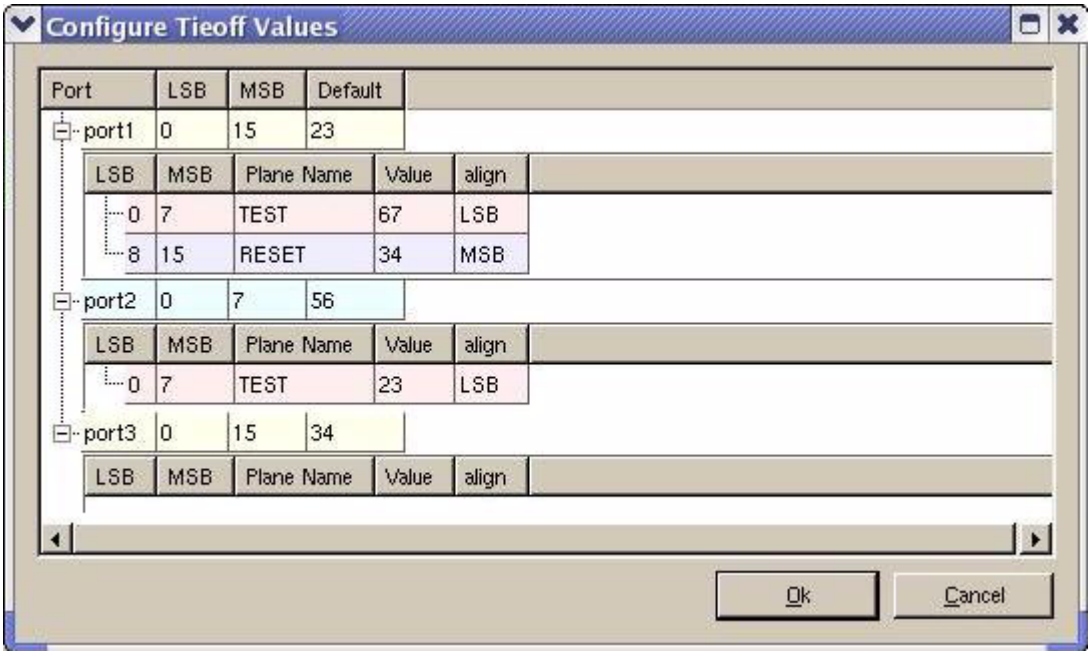
In this dialog, you can modify the tie-off value (per port) and create or remove tie-off connections. You can specify the LSB, MSB, Plane Name, Tie-off value, and alignment for each port.

The plane_name can take the following values:

BUS, CLOCK, EVENT, RESET, TEST, IO, POWER, DFT, DEBUG, MEMORY, SECURITY, VERIF

Click the OK button to update the tie-off connections for the instance.

Figure 5-1 Reconfigure Tieoff Connection



Interface Connections

Interface refers to a group of ports in an instance. When each port in an interface of a component instance is connected to the ports in another interface of a component instance, and when both the interfaces are instantiations of the same interface library, the connection is called an Interface connection.

If the CLASS attribute is present on an interface, one interface should be MASTER and the other one should be a SLAVE.

You can specify the src_instance, src_interface, dst_instance, dst_interface, dst_instance, dst_splice, src_splice, splice_tieoff, align, and plane_name in the Connections -> Interface table.

While creating an interface connection for a component with the -allow_create option set to Y, if the required interface does not exist, an interface gets created. In this case, if the other side of the interface is connected to a component that has the -allow_create option set to N, the interface library information is extracted from that side and a mirrored interface is

created on the component, which has the -allow_create option set to Y. Otherwise, a dummy interface gets created. A dummy interface is an interface without interface library information.

If the connection is made using the add_connection Tcl command and you have specified a library for the interface using the -interface_ref option, an interface will be created on the component using that interface library, if required. For more details, refer to *GenSys Tcl Commands Reference Guide*.

See Also

- Creating Interfaces
- Adding Ports On Interfaces
- Creating Interface Instances
- Mapping Interfaces
- Creating Connections

Unconnected

Unconnected refers to the ports of an instance that are intentionally left open in a design. Such ports are specified as open using the add_connection Tcl command with -noconn option.

Exported Interface

Exported interface refers to an interface that is created on a design to connect the ports of an interface of a component instance inside a design to the ports of that design. The ports of the exported interface are identified using a string called uniquiefier.

Default Open

Default open refers to the ports of an instance that are intentionally left open in a design. Such ports are created as open ports using the add_ports Tcl command with -open option set as TRUE.

Default Tieoff

Default tieoff refers to the ports of an instance that are connected to a default tieoff value. Such ports are created using the add_ports Tcl command with -default_value option set to a default value.

Understanding Connection Precedence

The purpose of connection precedence is to deal with situations where more than one connection specification attempts to make a connection to a pin. Precedence simply asserts that:

- Connections are made in a well-defined order.
- If an output pin is already connected, further attempts to connect to that pin are ignored.

Connection precedence is applied to input ports because input ports with multiple fanout and only single fanin is supported.

Let us look at an example of how connection precedence is applied using an interface connection, where we choose to connect a couple of bits of one signal in the interface using an ad-hoc connection.

Ad-hoc connections are made first, before any other connections. Now when we connect the interface, we skip the signals which are already connected and simply connect the remaining signals in the interface.

Following is the connection precedence order:

1. Unconnected
2. Adhoc connections
3. Tieoff connections
4. Interface connections: Interface connections are further given precedence in the following order:
   - Splice interface connections
   - Normal interface connections
   - Splice tieoff connections
5. Default tieoff connections
6. Default open connections

Validating Ad-hoc and Interface Connections

In case of ad-hoc and interface connections, validity of a connection can be checked using the validate_using attribute. For each pin/port of an instance or interface, a set of attributes can be defined. If the validate_using attribute is defined, the value of this attribute is used to check the validity of connection created from/to that pin/port.

The validate_using attribute can take a list of comma-separated string values. A connection is considered valid if any of the following conditions are satisfied:

- validate_using attribute is defined for both source and destination pins and at least one string in the value of this attribute is same in both.

  For example, validate_using = A,B,C for P1 and validate_using = A,D,F for P2. In this case, string A matches in the validate_using attribute of both P1 and P2. Therefore, the connection is valid.

- validate_using attribute is not defined for both source and destination pin. In this case, connection is valid because it is not required to be validated.

- validate_using attribute is defined for either source or destination pin and not defined for the pin at the other end of connection. In this case, connection is valid because it is not required to be validated for the pin for which the attribute is not defined.

If a connection is not valid, it will appear in red color in the Connections table.

Note:
In case of interface connections, if both the source and destination interfaces belong to the same interface library, the validation_using attribute is not considered to check the validity of connection.

Creating Connections

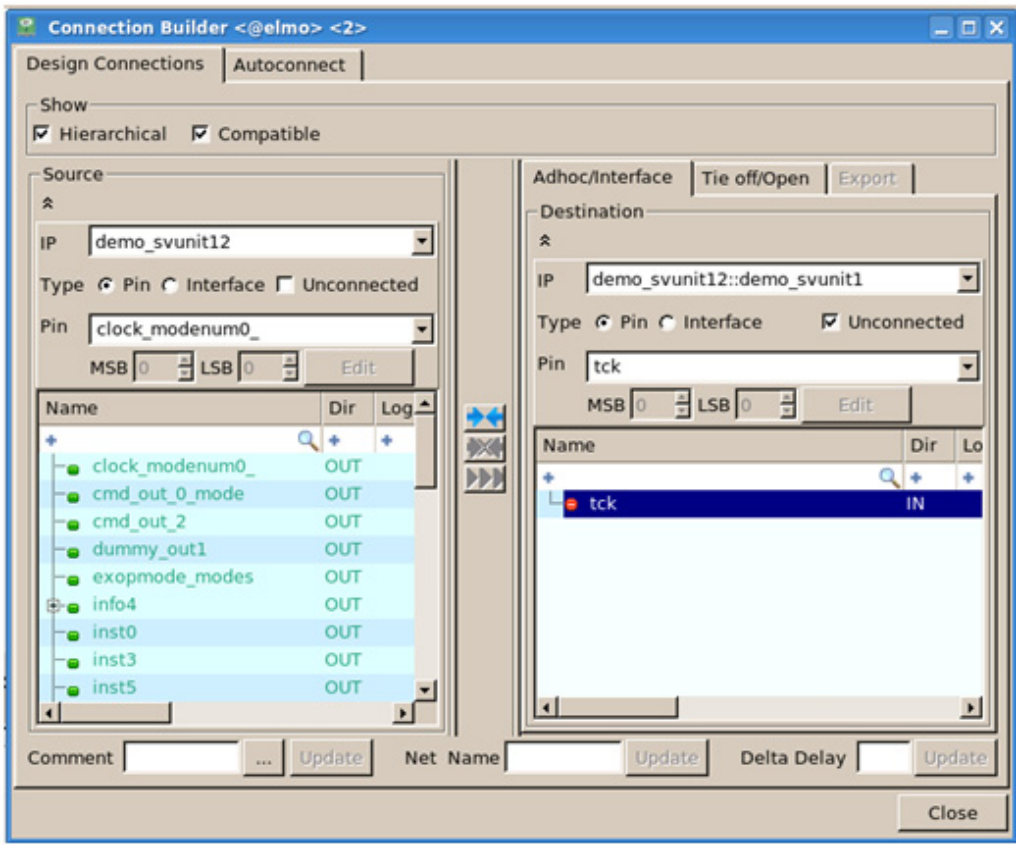
You can create connections in any of the following ways:

- Using the Connection Builder Dialog to Create Connections
- Using the Autoconnect Feature to Create Connections
- Using the Canvas View to Create Connections

Using the Connection Builder Dialog to Create Connections

The following figure shows the Connection Builder dialog:

Figure 5-2 Connection Builder Dialog



In the above dialog, the source connection details appear in the Source section and the destination connection details appear in the Destination section. Different types of connections are shown in different colors. For example, green appears for connected, red for unconnected, and yellow for partially connected.

Invoking the Connection Builder Dialog

To open the Connection Builder dialog, select a design and perform any of the following actions:

- Select the Tools->Connection Builder menu option.
- Click the

- button in the toolbar.

- Use the *Ctrl*+*Q* keyboard shortcut
- Right click the Connections table of the active design and select the Connection Builder option from the shortcut menu.

  Note:
  When a new design is loaded or when a new connection is created outside the Connection Builder dialog, you must elaborate the design before opening the Connection Builder dialog.

Details of the Connection Builder Dialog

The Connection Builder dialog contains the following tabs:

Design Connections Tab

Under this tab, when you select an instance from the IP drop-down list in the source or destination section, names of pins/interfaces of the that instance appear in the table at the bottom of that section. These pins/interfaces names appear as a complete hierarchical path from the top-level design.

You can hide or unhide the columns of this table by right-clicking the table header and selecting the required menu option.

The following check boxes are present under this tab:

Tab	Description
Hierarchical	Select this check box to view IPs within the hierarchy of the top-level IPs in the IP drop-down list of the Connections Builder dialog. By default, only names of all top-level IPs in a design appear in the IP drop-down list.
Unconnected	Select this check box to view only partially connected or unconnected pins in the source and/or destination sides.
Compatible	Select this check box to display only those pins/interfaces (in the Destination section) with which a connection is possible with the pin/interface selected in the Source section of the Connections Builder dialog.

Autoconnect Tab

Under this tab, you can specify details to generate connections automatically. For details, see Creating and Modifying Rules in the Connections Builder Dialog.

Creating a New Connection

To connect a pin/port through the Connection Builder dialog, perform the following steps:

1. Click the Design Connections tab.
2. Select a source instance from the IP drop-down list in the Source section.
3. Select a destination instance from the IP drop-down in the Destination section. You can also select multiple destination pins.

   If the selected destination instance has the allow_create property set to Y and a user-defined subroutine that returns a suggested pin name is registered, the button

   appears on the right side of the Pin drop-down list. Click this button to create a new pin with the suggested pin name in the Pin combo box. This pin name is returned by the registered subroutine.

   Note:
   You can create your own subroutine to suggest a pin name and register it with GenSys using the Perl API, registerDefaultAllowPortName. For more details, see *GenSys Customization Guide*.

4. Select an unconnected source pin/interface for that instance from the table in the Source tab.
5. Select a destination pin/interface of that IP with which you want to create a connection.
6. Specify the name of the net for the connection to be generated in the Net Name textbox.
7. Click the Connect button ()

   to create a connection.

A new connection appears between the source and destination IP and the tables in both Source and Destination sections show the following details:

Column	Description
Name	Specifies the name of Port/interface pin

Column	Description
Dir	Specifies the direction of Port/interface pin. The direction can be any of IN | OUT | INOUT.
Logical Port/ BusDefNam e	For interface ports, specifies the logical port to which the actual port/pin is mapped, if exists. For other ports it remains blank.
Connected To	Specifies the destination for the connected ports. For a top-level vector bus, this column shows the following values: <destination-name>, if all the bits are connected to the same destination. Partial, if any of the bit is left unconnected or default open or tied-off to a default value. Full, in the following cases: - If all the bits are connected but not to same destination - If a port in a fanout connection is connected to more than one port. - Any bit has status as open/temporary open - Any bit has status as tieoff/temporary tieoff Not connected, if the port is not connected.
Type	Specifies the type of connection. The type can be any of the following: Adhoc: Specifies that there is an adhoc connection between a source and destination pin.Interface: Specifies that there is a connection between a source and destination interface. Multi Adhoc: Specifies that there are multiple adhoc connections for the source output pin of an interface. Multi Interface: Specifies that there are multiple interface connections for the source output pin/port of an interface. Mix Adhoc/Interface: Specifies that there are adhoc and interface connections for the source interface.

To refresh the connection status for an interface or vector port, right-click the interface/vector port and select the Refresh option from the shortcut menu. This option is disabled for pin bits.

Note:
Please note the following points:

- GenSys enables you to view the details of all the redundant adhoc connections by using the execute_query Tcl command. Refer to *GenSys Tcl Commands Reference* manual for more details on execute_query Tcl command.
- If you try to connect an input pin that is already connected, GenSys prompts you to confirm the action. Three options are displayed: Override, Continue, and Abort. With Override, GenSys deletes the existing connection and recreates the new connection. With Continue, GenSys creates the new connection without removing the existing connection. In this case, you can later on specify which connection you want to consider, using the Overridden Connection Selector dialog. With Abort, GenSys does not proceed further to create the connection.
- If you try to connect an INOUT pin that is already connected with an OUT pin or IN port (of design), GenSys prompts you to confirm the action, as described above.

Specifying an Open and Tieoff Connection

The Connection Builder dialog allows you to connect an INPUT/INOUT port with a tieoff value, or you can leave an OUTPUT/INOUT port as open. You can configure these settings in the Tieoff or Open tab that appears in the Destination section.

Note:
If an interface is configured as open, the connection status of its logical ports is not updated. To view the updated status, you need to refresh the interface by using the shortcut menu.

Creating New Ports/Interfaces While Creating Connection

While creating a connection, you can create new ports/interfaces on master components and designs for which the allow_create option is set to Y. When you select the name of such component/design from the IP combo box, the PIN/I/face combobox becomes editable. You can then specify the name of the port or interface to be created.

Modifying MSB and LSB Values

You can also change the MSB and LSB values in the MSB and LSB combo boxes, respectively. The MSB-LSB range is validated and you cannot specify LSB higher than MSB or MSB lower than LSB.

Note:
For scalar signals, the MSB and LSB fields are disabled in both Source and Destination sections.

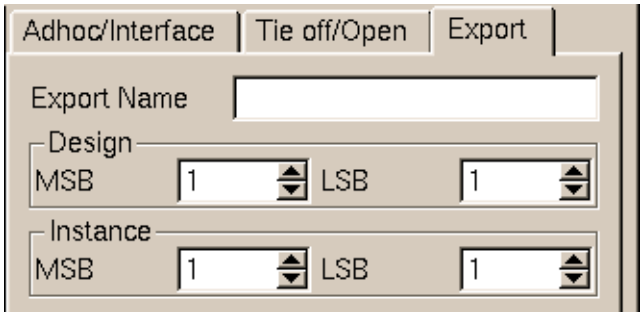
Modifying the Port Size

You can edit the port size for the components for which the allow_create option is set to Y. To change the size of such ports, select the port from the IP combo box and click Edit. The Edit MSB/LSB dialog appears. Enter the MSB and LSB values for the selected port in the MSB and LSB combo boxes, respectively, and then click OK.

Exporting an Instance Pin/ Interface Port

You can export the port/pin of an instance to a design.

For example, to export a pin of an instance, click the Export tab in the Destination section of the Connection Builder dialog. The following page appears under the Export tab:



Specify a name for the exported pin. You can also specify MSB and LSB values for the pin.

Based on the details specified, GenSys exports the pin/port and creates an adhoc connection between the instance pin and the generated design port.

Expanding or Collapsing Ports/Interfaces

The shortcut menu in the source and destination details table provide the following options to view or hide the details of vector pins/ports/interfaces:

- Expand: Shows the details of the selected pin/port/interface.
- Expand All: Shows the details of all vector pins/ports/interfaces.
- Expand Interfaces: Shows the details of all interfaces.
- Collapse All: Shows the details of all vector pins/ports/interfaces.
- Collapse Interfaces: Hides the details of all interfaces.

Elaborating Connections of a Design

To view the exact ports that are connected, you should elaborate connections.

To elaborate connections of an active design, select an instance and click the Elaborate button. After elaboration, the connection between the individual pins/ports of the instances are displayed. In addition to creating adhoc connections for interface connections, elaboration also resolves the parameters. For example, if a parameter is used in MSB of a port, any change in the value of the parameter is reflected only after elaboration.

Note:

Elaboration happens automatically during RTL generation and RTL Health Check generation.

Modifying an Existing Connection

To change the connection details of connected pins, perform the following steps:

1. Select the connected pin/port under the Source section.
2. Select the new destination instance from the IP drop-down list under the Destination section.

   The corresponding pin/interface details appear in the table. Alternatively, click Show Connections button (

) to load the destination details.
3. Change the destination pin/interface details.
4. Click the Connect button (

5.) to change the connection.

If a source pin is connected with multiple pins, and you click the

, a list of destination pin connections with that source pin gets displayed in the Connected To column of the source pin. By default, the destination details of the first connection with that source pin is loaded in the Destination section. When you again click

, the destination details of the next pin connection with the source pin gets loaded, and so on.

Updating the Delta Delay Value

You can also specify a delay value (in nano seconds) for a connection in the Delta Delay text box and then click Update. This delay value for connection can be dumped to RTL dump. You can also provide a description for the connection in the Comment text box and then click Update. The Update button is enabled only when both source and destination of a connection are loaded and selected in the Connection Builder dialog.

Note:
In case of tieoff and open connections, Delta Delay text box is disabled.

Deleting a Connection

To delete an existing connection, select the source and destination of that connection. To select the destination of a connection, perform the following steps in the given sequence:

1. For a pin-to-pin/pin-to-port/port-to-pin connection, click on the

2. button. It will highlight the destination pin(s)/port(s). This will enable the Disconnect button (

3.).
4. For a tieoff/temporary tieoff, select the Tie off tab (that appears for instance input pin/ design output port) or the Tie off/Open tab (that appears for INOUT pin/port) in the Destination section. This will enable the

5. button.
6. For an open/temporary open connection, select the Open tab (that appears for instance output pin /design input port) or Tie off/Open tab (that appears for INOUT pin/port) in the Destination section. This will enable the

7. button.
8. Click on the

9. button to delete the connection. The disconnected pins/ports will be displayed as Not Connected under the Design Connections tab of the Connection Builder dialog.

Using the Autoconnect Feature to Create Connections

While working on SoC designs, you instantiate various IPs and create connections between instances. However, the total number of connections to be created is generally huge. So creating all connections manually can be very time-consuming and may increase time-to-market.

To solve this problem, you can use the autoconnect feature to generate connections automatically in GenSys in any of the following ways:

- Generating Automatic Connections by Using Rules
- Generating Automatic Connections by Using Tcl Commands

Generating Automatic Connections by Using Rules

A rule specifies an infrastructure that captures user-intent to generate connections automatically. For example, you may create a rule to connect the Clk1 pin of the source instance PLL with all the destination instance pins that match the regular expression .*clk.*.

Please note the following points:

- This feature does not consider pins/interfaces that are already connected. The specified rules are applicable only for unconnected pins/interfaces.
- This feature does not connect pins that might lead to bad connectivity, such as connecting IN-IN, OUT-OUT pins.

Once a rule is run, GenSys generates the AutoconnectReport.log file in the current working directory. This file contains the following information:

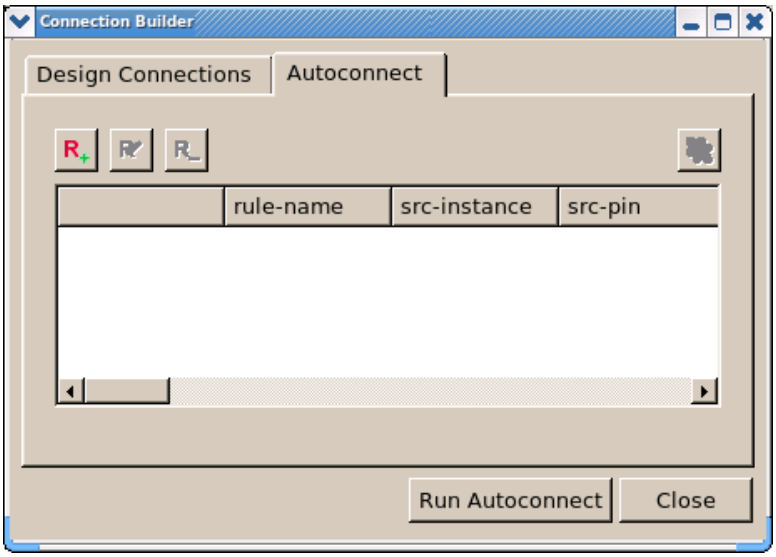
- Connections created successfully and a warning generated, if any while making a connection.
- Connections that could not be created and the corresponding reason, such as invalid direction of pins.

Creating and Modifying Rules in the Connections Builder Dialog

You can create and modify rules under the Autoconnect tab of the Connections Builder dialog.

The following figure shows the page under the Autoconnect tab:

Figure 5-3 Autoconnect Tab in the Connections Builder Dialog



The above page contains a table in which each row represents a rule. This table is same as the Autoconnect table of the currently loaded design.

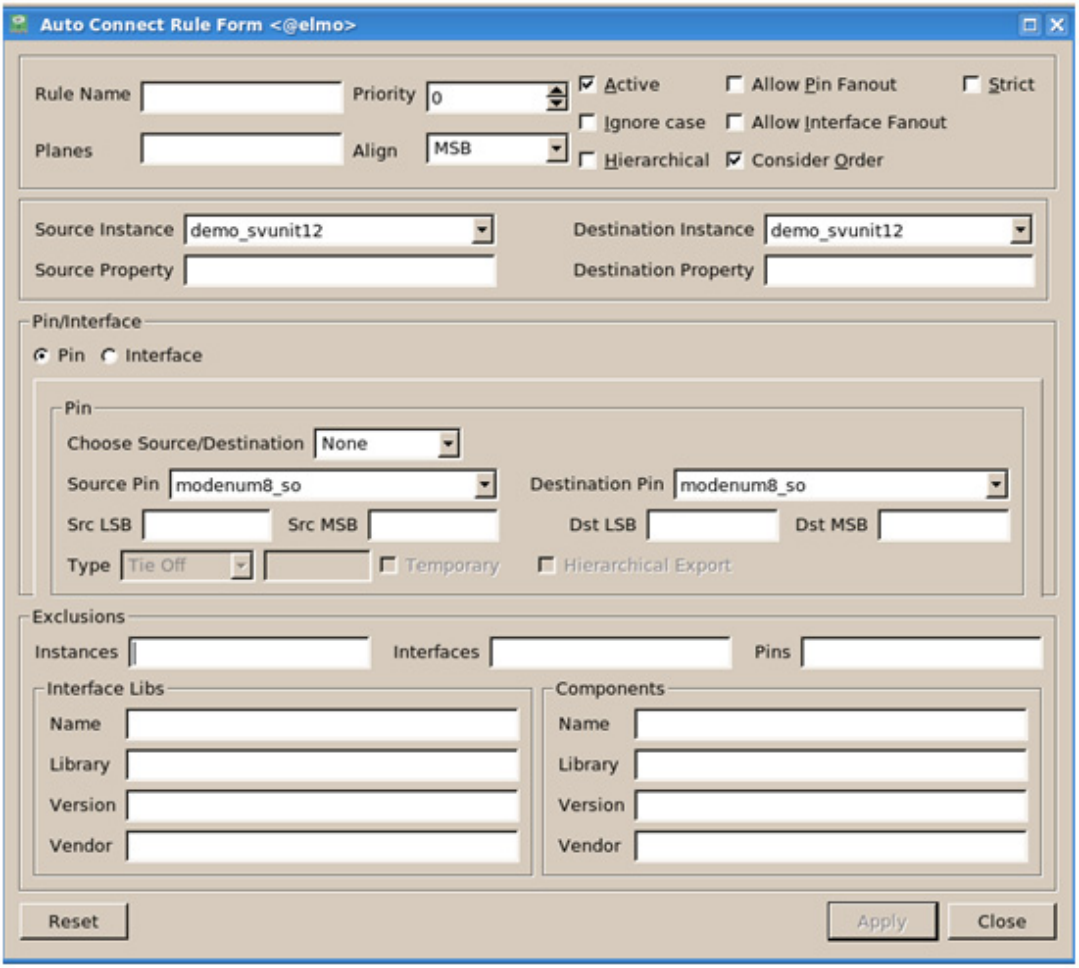
Adding Rules

To add a new rule, perform the following steps:

1. Click the Add Rule button (

2.).

   The Auto Connect Rule Form dialog appears, as shown in the following figure:



3. In the above dialog, specify the rule details.

   - Source and destination instance: Enter or select the name of the source and destination instance from the *Source Instance* and *Destination Instance* drop-down lists.

   Based on the selected instance, the *Source Pin* and *Destination Pin* drop-down list is populated.

   If the instance name is invalid (this may occur in case of a regular expression), the corresponding pin list will be empty. If the instance name is left blank, the corresponding pin list will show the design port list.

- Allow Pin Fanout: Default is unchecked. Earlier the Autoconnect feature use to create fan-out connections by default, but this release onwards to create fan-out connections, you need to select this checkbox.
- Allow Interface Fanout: Default is unchecked, and an interface is allowed to drive only one interface. Select this checkbox to enable an interface to drive multiple interfaces. Earlier Autoconnect did not allow interface fan-out connections. Previously this setting was not possible through the Autoconnect (AC) widget, but the user could specify this using the set_autoconnect_options Tcl command. This release onwards we have added this checkbox per rule basis and the user can enable/disable it for particular rule from the AC widget itself.
- Hierarchical: When this checkbox is checked, the *Source Instance* and *Destination Instance* drop-down lists show the hierarchical paths for all the existing instances in the design recursively.

  If it is unchecked, the instance lists show only the instances of the first level of the currently active design.
- Consider Order: Default is selected. This means that ordering is inferred out of regular expressions, such as m[a|b|c] or a[0-5]. With this release onwards by default ordering is inferred out of such regular expressions until it set to false.

4. Click the OK button.

After performing the above steps, the above dialog closes and a new row appears for the newly created rule under the Autoconnect tab.

Modifying Rules

To modify an existing rule, perform the following steps:

1. Select the required row in the table under the Autoconnect tab.

   The Auto Connect Rule Form dialog appears in which details of the selected rule appear in appropriate textboxes.
2. Perform the required updates in this dialog.
3. Click the OK button.

After performing the above steps, the specified changes appear in the row of the selected rule.

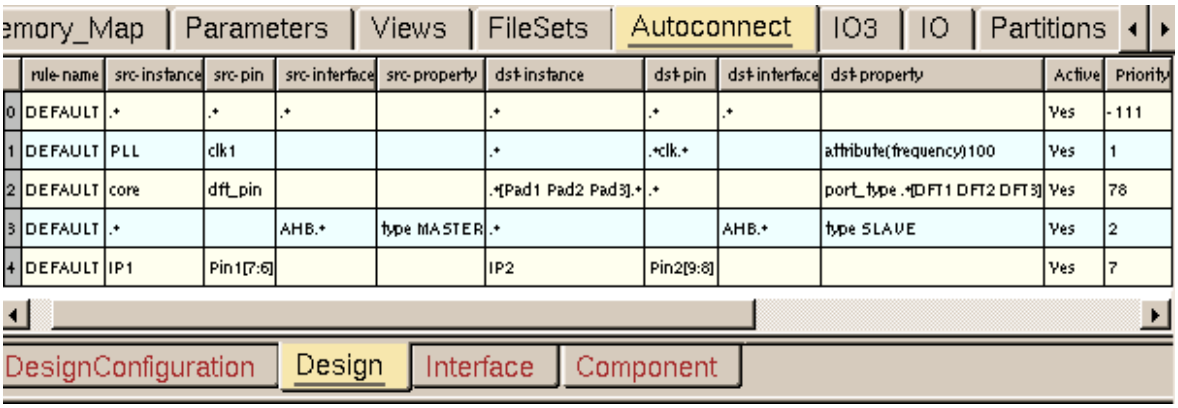
Deleting a Rule

To delete a rule, select the row of that rule and click the Delete Rule button.

Creating and modifying rules through tcl commands

- Adding Rules: New rules can be created using the add_autconnect_rule Tcl command
- Modifying Rules: Existing rules can be modified/updated using the add_autoconnect_rule Tcl command with the -update argument.
- Deleting Rules: Existing rules can be removed using the delete_autoconnect_rule Tcl command.

After you have added/modified the rules, you can view them in the Autoconnect table, as shown in the following image. This view is read-only.



	rule-name	src-instance	src-pin	src-interface	src-property	dst-instance	dst-pin	dst-interface	dst-property	Active	Priority
0	DEFAULT	.*	.*	.*		.*	.*	.*		Yes	-111
1	DEFAULT	PLL	clk1			.*	.*clk.*		attribute(frequency)100	Yes	1
2	DEFAULT	core	dft_pin			.*[Pad1 Pad2 Pad3].*	.*		port_type .*[DFT1 DFT2 DFT3]	Yes	78
3	DEFAULT	.*		AHB.*	type MASTER	.*		AHB.*	type SLAVE	Yes	2
4	DEFAULT	IP1	Pin1[7:6]			IP2	Pin2[9:8]			Yes	7

Running the Specified Rules

After creating or modifying rules, run the rules by performing any of the following actions:

- Click the Run AutoConnect button in the Connection Builder dialog to run the rules.

  Note:
  If you do not want to elaborate the design before running rules, deselect the Elaborate before AutoConnect checkbox.

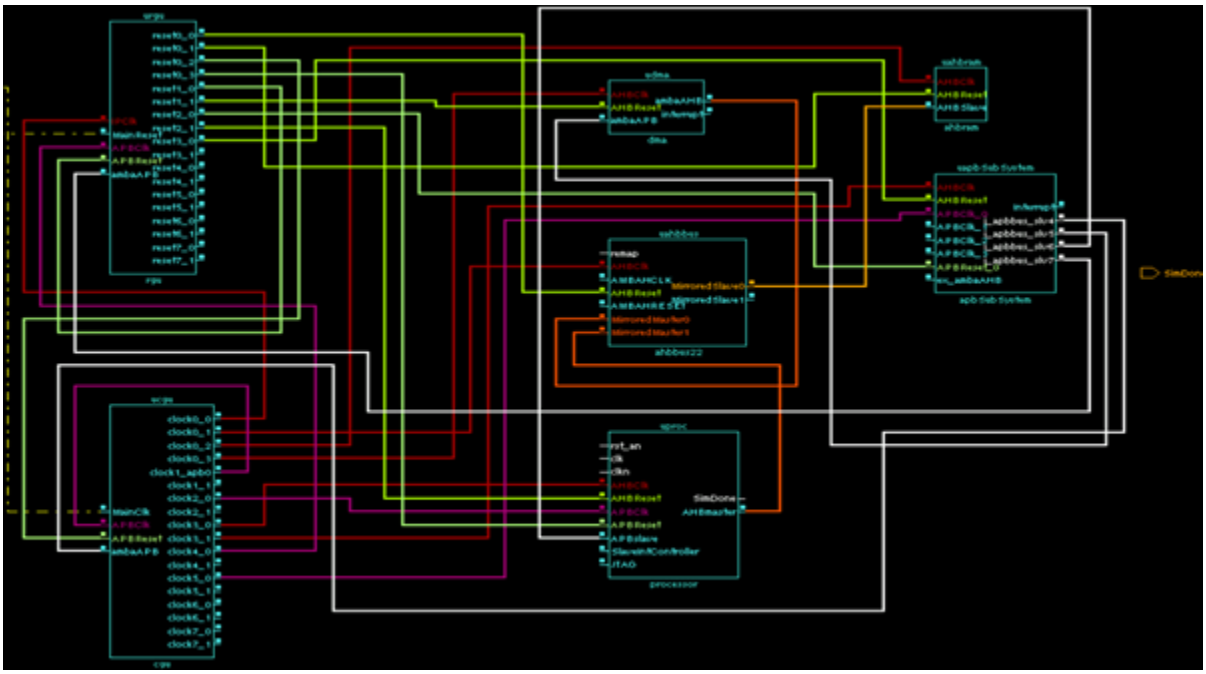
- Run the run_autoconnect Tcl command.

Once the rule run is complete, GenSys generates the AutoconnectReport.log in the current working directory. This file contains the following information:

- Connections created successfully and a warning generated, if any, while making a connection.
- Connections that could not be created and the corresponding reason, such as invalid direction of pins, etc.

Generating Automatic Connections by Using Tcl Commands

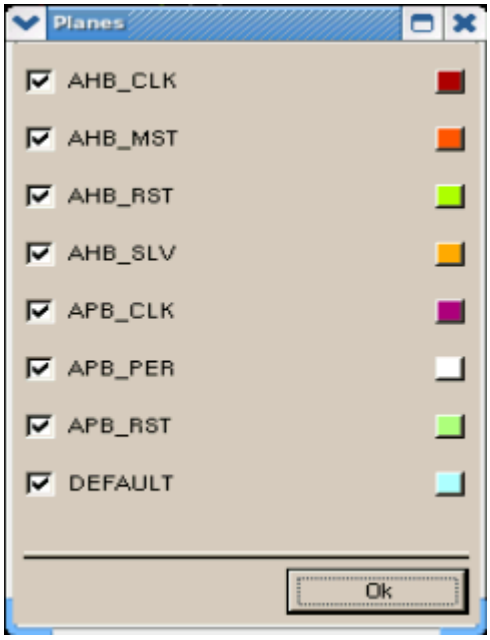
Consider the following SoC:



The following is the architecture defined for the above SoC:

- rgu: reset generator unit
- cgu: clock generator unit
- uapbSubSystem: sub system which APB peripherals and AHB to APB protocol conversion
- udma, uproc: As AHB masters IP,
- uahbram: As AHB slave and uahbbus - switch/channel for AHB masters and AHB slaves.

Now consider that you want to run some autoconnect rules on this SoC so that the following planes are generated:



The planes available are:

- AHB_CLK and AHB_RST: For AHB clock and reset network
- APB_CLK and APB_RST: For APB clock and reset network
- AHB_MST and AHB_SLV: For AHB master and slave network
- APB_PER: For APB peripherals connectivity network

Making Rule-Based Automatic Interface Connections

Use the following Tcl commands to make rule-based automatic interface connections in this case:

```
add_autoconnect_rule -name AHB_MSTs -src_inst {^u(d|p)} -src_intf {AHB}
-dst_inst uahbbus -dst_intf {^Mir} -priority 6 -plane AHB_MST

add_autoconnect_rule -name AHB_SLVs -src_inst {uahbbus} -src_intf {^Mir}
-dst_inst {(ram|Sys)} -dst_intf {AHB} -priority 7 -plane AHB_SLV

add_autoconnect_rule -name APB_PERs -src_inst {.*Sys} -src_intf {slv}
-dst_inst {.*} -dst_intf {APB} -priority 8 -plane APB_PER -exclude_inst
{uahbbus}

run_autoconnect
```

Making Rule-Based Automatic Pin Connections

Use the following Tcl commands to make rule-based automatic pin connections in this case:

```
add_autoconnect_rule -name CLK_MAIN -src_pin {^clkin$} -dst_inst ucgu
-dst_pin clkin -plane MAIN_CLK
add_autoconnect_rule -name RST_MAIN -src_pin rstin_an -dst_inst urgu
-dst_pin rstin_an -plane MAIN_RST
add_autoconnect_rule -name APB_ADHOC -src_inst {uapbSubSystem} -src_pin
{.*slv7.*} -dst_inst {udma} -dst_pin {^p.*} -priority 8 -plane
APB_ADHOC_PLN
```

Making Connections Based on CONNECT_ORDER

Use the following Tcl commands to make connections based on CONNECT_ORDER, as per your preference:

For Interfaces

```
select_interfaceinst -instance udma -interface ambaAHB
add_attribute -name CONNECT_ORDER -value 1
select_interfaceinst -instance uahbbus -interface MirroredMaster0
add_attribute -name CONNECT_ORDER -value 1
select_interfaceinst -instance uahbbus -interface MirroredMaster1
add_attribute -name CONNECT_ORDER -value 2
select_interfaceinst -instance uproc -interface AHBmaster
add_attribute -name CONNECT_ORDER -value 2
add_autoconnect_rule -name AHB_MSTs -src_inst {^u(d|p)} -src_intf {AHB}
-dst_inst uahbbus -dst_intf {^Mir} -priority 6 -plane AHB_MST
```

For Pins

Note:

It is not recommended to use connect order for pins because instead of setting CONNECT_ORDER on pins, it is better to make connections directly. However, it is fine in case of limited use of connect order for pins.

```
select_terminal -instance uapbSubSystem -pin i_apbbus_slv7_paddr
add_attribute -name CONNECT_ORDER -value 0
select_terminal -instance uapbSubSystem -pin i_apbbus_slv7_psel
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance uapbSubSystem -pin i_apbbus_slv7_pwrite
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance uapbSubSystem -pin i_apbbus_slv7_pwdata
add_attribute -name CONNECT_ORDER -value 3
select_terminal -instance uapbSubSystem -pin i_apbbus_slv7_penable
add_attribute -name CONNECT_ORDER -value 4
select_terminal -instance uapbSubSystem -pin i_apbbus_slv7_prdata
add_attribute -name CONNECT_ORDER -value 5
select_terminal -instance udma -pin paddr
add_attribute -name CONNECT_ORDER -value 0
select_terminal -instance udma -pin psel
```

```
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance udma -pin pwrite
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance udma -pin pwdata
add_attribute -name CONNECT_ORDER -value 3
select_terminal -instance udma -pin penable
add_attribute -name CONNECT_ORDER -value 4
select_terminal -instance udma -pin prdata
add_attribute -name CONNECT_ORDER -value 5
add_autoconnect_rule -name APB_ADHOC -src_inst {uapbSubSystem} -src_pin
{.*slv7.*} -dst_inst {udma} -dst_pin {^p.*} -priority 8 -plane
APB_ADHOC_PLN
run_autoconnect
```

Making Sub-Hierarchy Interface Export and Making Connections with it

Use the following Tcl commands to make sub-hierarchy interface export and make connections with it:

```
add_autoconnect_rule -name APB_SUB_EXPORT -src_inst
{uapbSubSystem::i_apbbus8} -src_intf {^Mirr.*Slave(4|5)} -export yes
run_autoconnect -name APB_SUB_EXPORT

#now connecting between the xported and the existing interfaces
select_interfaceinst -instance uapbSubSystem -interface MirroredSlave4
add_attribute -name CONNECT_ORDER -value 0
select_interfaceinst -instance uapbSubSystem -interface MirroredSlave5
add_attribute -name CONNECT_ORDER -value 1
select_interfaceinst -instance ucgu -interface ambaAPB
add_attribute -name CONNECT_ORDER -value 1
select_interfaceinst -instance urgu -interface ambaAPB
add_attribute -name CONNECT_ORDER -value 0
add_autoconnect_rule -name APB_MIRROR_SLAVES -src_inst {uapbSubSystem}
-src_intf {^Mirr.*Slave(4|5)} -dst_inst {^u.*gu} -dst_intf {^ambaAPB$}
run_autoconnect -name APB_MIRROR_SLAVES
```

Chaining Connections

Use the following Tcl command for chaining connections:

```
add_autoconnect_rule -name PGOOD_IN -src_inst {p.*} -src_pin
{^PGOOD_IN.*M.*} -dst_inst {p.*} -dst_pin {^PGOOD_OUT.*M.*} -plane
PGOOD_IN
```

Making Connections Based on Properties

Use the following Tcl commands to make connections based on properties:

```
add_autoconnect_rule -name SCAN_CHAIN -src_inst {p.*} -src_pin
{.*SCAN.*IN} -src_property "port_type SCANIN" -dst_inst {p.*} -dst_pin
{.*SCAN.*OUT} -dst_property "port_type SCANOUT"
run_autoconnect
```

Making Connections Based on Regular Expression Capturing

This enhancement has been done to allow users to make their rules more expressive by allowing the source and destination regular expressions to influence each other. To achieve this, we have used the same convention as Perl/Qt regular expressions wherein regexps within brackets are stored within some variables. This process is known commonly as capturing where we can group part of the regular expression together. For example, a regular expression like "inst(.*)_pin(.*)" when matches an identifier, such as "inst1_pin2", will also provide the variables "$1 = 1" and "$2 = 2". For example:

```
add_autoconnect_rule -name RULE1 -src_inst cu_xxx0 -src_pin
{sd_tclm(\S+)} -dst_inst cu_yyy0 -dst_pin {sd_tclm$1} -priority 1
```

This allows a user to refine his search space for the connection destination based on regular expression groupings in the source side. In the above case, the grouping "\S+" is captured as $1, which is used to resolve the destination.

What the above autoconnect rule enables is the following kind of connection :

```
cu_xxx0.sd_tclmxxx00 > cu_yyy0.sd_tclmxxx00
cu_xxx0.sd_tclmxxx01 >  cu_yyy0.sd_tclmxxx01
```

Without this kind of rule expressiveness, we would have used "sd_tclm.*" in the current infrastructure and that would have given us a multiple source/multiple sink behavior and we might have ended up with "cu_xxx0.sd_tclmxxx00 ' cu_yyy0.sd_tclmyyy01" kind of connection.

We have added capturing support for regular expressions of -src_inst, -src_pin, -src_lsb, -src_msb, -src_intf , -dst_inst, -dst_pin, -dst_lsb, -dst_msb, -dst_intf switches. Since the variable name is dependent on the order of capturing, the order as provided by the add_autoconnect_rule Tcl command is used. The variable names follows the same order from left to right as provided in the original Tcl command. For example:

```
add_autoconnect_rule -name RULE2 -src_inst (.*)_xxx0 -src_pin
{(.*)_tclm(\S+)} -dst_inst $1_yyy0 -dst_pin {$2_tclm$3} -priority 1
```

This Tcl command implies the following kind of connectivity :

```
cu0_xxx0.sdx_tclmxxx00 > cu0_yyy0.sdx_tclmxxx00
cu0_xxx0.sdx_tclmxxx01 >  cu0_yyy0.sdx_tclmxxx01
cu1_xxx0.sdy_tclmxxx00 > cu1_yyy0.sdy_tclmxxx00
cu1_xxx0.sdy_tclmxxx01 >  cu1_yyy0.sdy_tclmxxx01
```

We have made other enhancements to enable better rule expressiveness to provide the user more control over the scalar level connectivity. This means that the user can control which bits of a vector can be used for connectivity. Since the user might want to make the scalar splicing dynamic, we have added Perl formulas in the msb/lsb selection for vectors. To achieve this we have added new switches -src_lsb, -src_msb, -dst_lsb,and -dst_msb.

Note:

We have deprecated the old format of specifying lsb/msb i.e pin[lsb:msb].

For example, the following autoconnect rule should imply:

```
add_autoconnect_rule -name RULE3 -src_inst {m[a|b|c]} -src_pin {int(\d+)}
-dst_inst {md} -dst_pin int -dst_msb {(2-$1)*2+1} -dst_lsb {(2-$1)*2}

ma.int00[1:0] > md.int[5:4]
ma.int01[1:0] > md.int[3:2]
ma.int02[1:0] > md.int[1:0]
```

The following guidelines describe how and where captured text can be used:

- Source side captured text can be used at any field in source side.
- Destination side captured text can be used at any field in the destination side.
- Source side captured text can be used at same field of destination side (src_inst > dst_inst, src_pin > dst_pin, src_intf > dst_intf)
- Destination side captured text can be used at the same field of source side (dst_inst > src_inst, dst_pin > src_pin, dst_intf > src_intf)
- For -src_lsb, -src_msb, -dst_lsb, and -dst_msb arguments captured text from any field can be used.
- Same captured text cannot be accessed at two fields across sides (e.g. $1 on src_pin and dst_pin both).

Making Connections Based on Regular Expression Ordering

The autoconnect rules have been enhanced so that if there is any order specified in the regular expressions, that order will be used to make connections in case of multiple source/sinks. For example:

```
add_autoconnect_rule -name RULE3 -src_inst {m[a|b|c]} -src_pin {int(\d+)}
-dst_inst {md} -dst_pin {int[5-0]}
```

This implies the following:

```
ma.int01[1] > md.int5
ma.int01[0] > md.int4
mb.int02[1] > md.int3
mb.int02[0] ' md.int2
mc.int03A ' md.int1
mc.int03B ' md.int0
```

regexp "m[a|b|c]" used in this rule implies that the tool will first match ma followed by mb followed by mc, in that order.

Another example to highlight this scenario is as follows:

```
add_autoconnect_rule -name RULE4 -src_inst {m[a|b|c]} -src_pin mclk
-dst_inst {m[d|e|f]} -dst_pin pclk
```

For the above autoconnect rule, this will resolve to a M sources and N sink scenario (where M = N = 3) and autoconnect will make the connections between ma and md, mb and me, mc and mf.

Using the Canvas View to Create Connections

For details, see Creating Direct Connections Between Objects and Creating Connections with Additional Details.

Using Tcl to Create Connections

Use the add_connection Tcl command to create connections. For details on this command, refer to the *Tcl Commands Reference Guide*.

Controlling Net Names while Creating Connections

While making connections, you can control the net names by using the set_netname_option Tcl command.

Also see Specifying a Net Name for a Connection.

Managing Overridden Connections

It may happen that multiple types of connections exist between two points in a design. Such connections are known as overridden connections.

For the overridden connections, GenSys considers the connection that has the highest priority. For details, see Understanding Connection Precedence.

Examples of Overridden Connections

Following are some examples of overriden connections which conflict which each other. At any given point of time, a design can have only one of these kinds of connections.

- A particular instance/terminal combination has Adhoc, Interface, Tieoff as well as an open connection. In the current use-model, GenSys picks up open connection as this

has the highest priority among all the four types of connections provided. Hence, this terminal shall remain open after elaboration or in RTL.

- In case of a multiple-driver situation, if you have two or more Adhoc connections, A1 and A2, on the same Instance/Terminal combination. In the current use-model, GenSys will pick the first one added by the user (A1) and ignores the second one (A2), after flagging a warning. Users should have a capability to choose A2 instead of A1. Using the Overriden Connection Selector dialog, you can select any of these connections.
- If the following connectivity has been provided for input terminals T1 and T2:

  1. Create an adhoc connection, A1, from T1 to T3
  2. Create interface connection, Inf1, from Interface of T1/T2 to Interface of T4.
  3. Delete the T1 and T2 splice from Inf1 using the Tcl command delete_connection (D1)
  4. Create an adhoc connection A2 from T2 to T4

  From the scalar terminal point of view, connections for terminal T1 are A1, Inf1 and D1 and connections for terminal T2 are Inft1, D1 and A2.

  Currently, the output has connection D1 for terminal T1 and A2 for terminal T2. That means, terminal T1 remains open while T2 is connected to T4.

  But, the important point to understand here is, that the user might have done delete connection for T2 mistakenly, and he just wanted to do the delete connection for T1 at step (3).

  User now has a capability to select between multiple overridden connections as well as delete connections for every scalar terminal. Hence the users should be given an option to choose between the following:

  - For terminal T1: A1, Inf1, D1
  - For terminal T2: Inf1, D1 and A2

  Depending on the selection, appropriate connectivity is generated.

  Note:
  In case of overlapping connections, no input is required from users. Such over-lapping connections get removed from the connectivity list after elaboration.

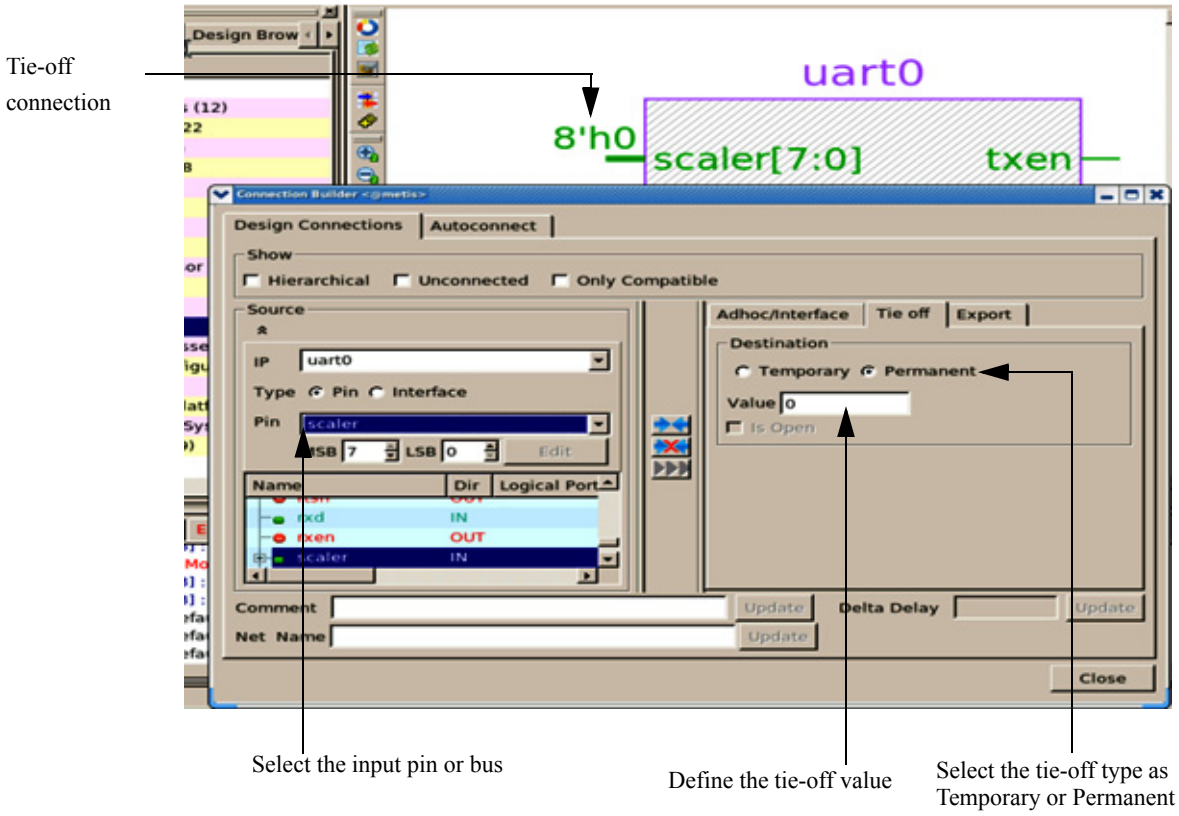
Generating Tie-off Connections

In tie-off connections, only an input pin can be tied-off to a constant value and only an output pin can be left open.

You can specify tie-off connections in a variety of formats, such as binary or hexadecimal format. For example, 0xA24, 10110, or 4’b1110.

Using GUI to Generate Tie-Off Connections

Use the *Connection Builder* dialog to specify tie-off connections, as shown in the following figure:



Using Tcl to Generate Tie-Off Connections

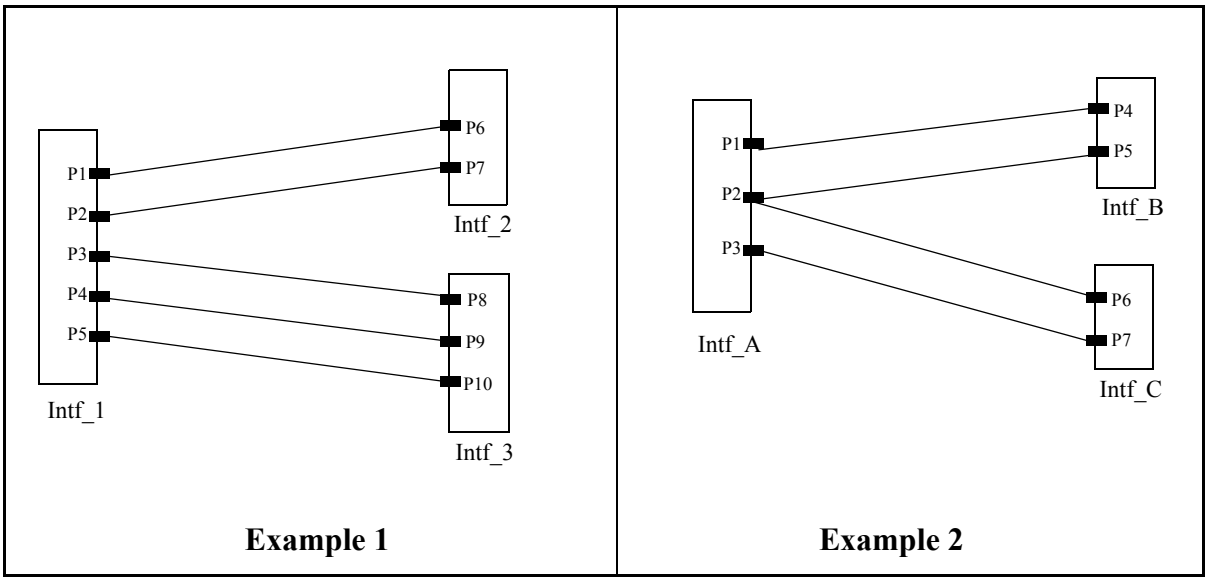
Use add_connection Tcl command to force tie-off or leave an output pin open. It is very similar to standard connections, but one side of the connection is different.

```
# Tie-offsadd_connection -instance pll -pin disable -tieoff 0
# Vector pin tied to constant valueadd_connection -instance pll -pin
haddr_s -tieoff 0xA24
# Vector pin tied to constant valueadd_connection -instance pll -pin
hburst_s -tieoff “10110”
```

Generating Splice Connections

In splice connections, pins of one interface are connected with the pins of multiple interfaces.

The following figures show the examples of splice connections:



To generate splice connections, use the -splice argument of the add_connection Tcl command. For details, refer to the *Specifying Splice Ports* topic of *GenSys Tcl Commands Reference Guide*.

Specifying Vector Port Alignment

Vector ports can be MSB aligned or LSB aligned based upon which connections are generated. You can specify this alignment by using the set_elab_option Tcl command.

For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

Specifying Bit Endianness

Bit endianness determines the direction of a connection. It specifies whether the MSB of a port is left-most (little endianness) or right-most (big endianness).

For example, consider the P1[3:0] port. In this case, if MSB is 3 (that is left-most), the bit endianness is big endianness. However, if the MSB is 0 (that is right-most), the bit endianness is little endianness.

You can specify a default bit endianness by using the set_lsb_bit Tcl command.

While creating a port, you can specify the bit endianness by using the -lsb_bit argument of the add_port Tcl command.

For details on these commands, refer to the *GenSys Tcl Commands Reference Guide*.

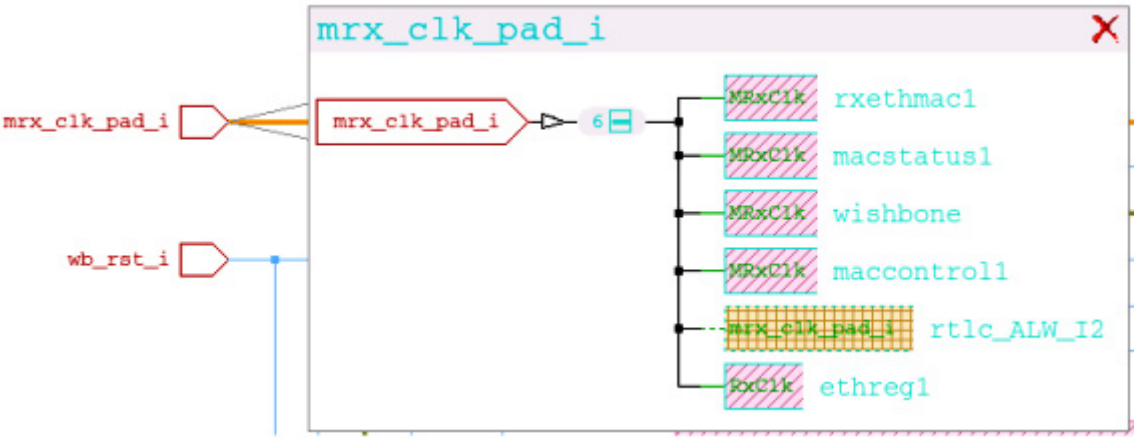
Debugging Connections in Schematic

You can debug connections in schematic by <Ctrl>+double-clicking on a pin or a pin bus to view the objects that pin or pin bus is driving.

For example, the following figure shows that mrx_clk_pad_i is driving six objects:

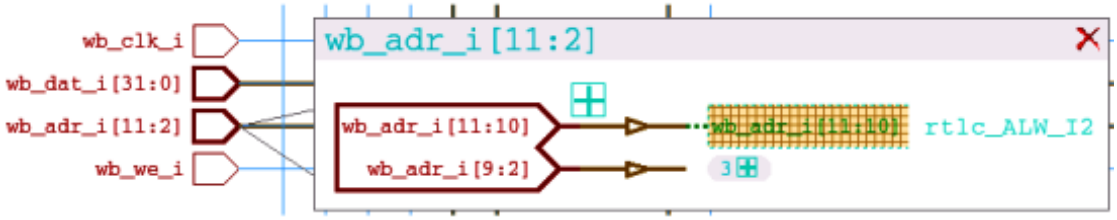
Figure 5-4 Debug connections in schematic

In the above figure, click the 6+ button to see the six objects that are driven by mrx_clk_pad_i, as shown in the following figure:



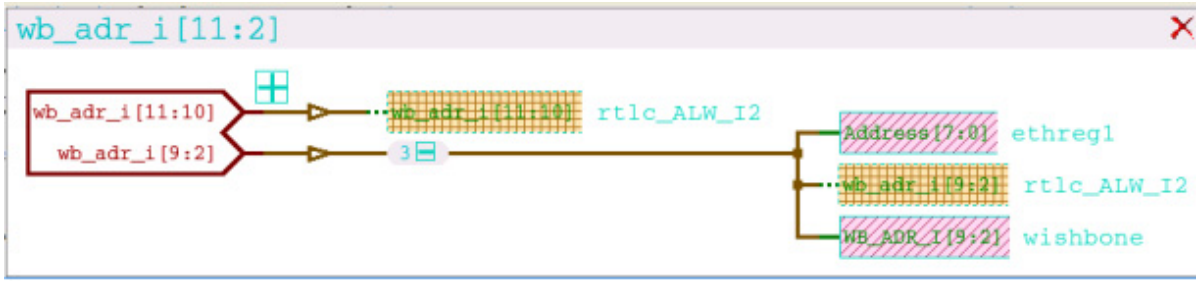
Similarly, you can debug connections through vector ports. For example, the following figure shows a vector port where some bits rip out to one instance and other bits rip out to other destinations:

Figure 5-5 Debug connections in schematic



In the above schematic, click the 3+ button to see the following:

Figure 5-6 Debugging connections through vector ports



Support for Connectivity Verification

This feature is useful for assembly flow where you can connect different IPs in a subsystem or SOC by add_connection or similar commands, or delete an incorrect connection. GenSys has been enhanced to generate SGDC constraints for connection verification. You can then use the generated SGDC file and the GenSys-generated RTL after assembly operations as an input for the SpyGlass Connectivity Verify product to verify the connections in the generated RTL.

For more information, see the following:

- Enable the Generation of SGDC Constraints
- Supported Connection Addition and Deletion Commands
- Type of SGDC Constraints Generated
- Example

Enable the Generation of SGDC Constraints

To enable the generation of SGDC constraints for connection verification, the following arguments have been added to the set_gensys_option Tcl command:

- -write_connectivity_check_sgdc_file: When this option is set to true, GenSys starts tagging each connection addition and deletion command executed in GenSys and internally creates SGDC constraints required to verify those connection addition and deletion commands.
- -connectivity_check_sgdc_filename: Use this argument to specify the name of the generated SGDC file. If the file name is not provided, GenSys writes the SGDC constraints in an internally generated file called verify_connections.sgdc. This file is located in the report directory.

The syntax is as follows:

```
set_gensys_options   -write_connectivity_check_sgdc_file true
-connectivity_check_sgdc_filename <file-name>
```

Supported Connection Addition and Deletion Commands

GenSys supports the following connection addition commands:

- add_connection
- add_hier_connection
- export_pin
- export_interface
- export_unconnected

These commands can also be specified by auto-connect rules. For all these commands, GenSys will generate a SGDC constraint so that actual connections can be verified in the GenSys-generated RTL.

Type of SGDC Constraints Generated

The following SGDC constraints can be generated:

- require_path: GenSys generates this SDGC constraint for simple or hierarchical pin-to-pin connections. In addition, this constraint is generated for interface connections for all ports and interfaces.

- require_value: GenSys generates this SDGC constraint for tie-off connnections. In addition, for hierarchical connections, the generated constraints contain the hierarchical path.
- illegal_path: GenSys generates this SDGC constraint for pin-to-pin connection deletion.
- illegal_value: GenSys generates this SDGC constraint for deletion of tie-off connection.

For the deletion of hierarchical connections, the generated SGDC constraint also contains the hierarchical path. However, for delete_connection if the corresponding add_connection was executed earlier in the flow and therefore a require_path or require_value SGDC constraint was added, then the delete_connection command will not generate any illegal_path or illegal_value SGDC constraint. Instead, it deletes the previous constraint.

Connection command	Type of command	Generated SGDC
add_connection -port SEF -lsb 0 -msb 1 -instance U1 -pin D -lsb 0 -msb 1	Simple pin-to-pin connection	require_path -from top.SEF[0] -to top.U1.D[0] -path_type direct require_path -from top.SEF[1] -to top.U1.D[1] -path_type direct
add_connection -instance I1 -pin P1 -instance dmass::DMA -pin P2	Hierarchical pin-to-pin connection	require_path -from top.I1.P1 -to top.dmass.DMA.P2
add_connection -instance I1 -pin P1 -tieoff 2	Simple tie-off connection	require_value -name top.I1.P1[1] -value 1 require_value -name top.I1.P1[0] -value 0
add_connection -instance dmass::DMA -pin P2 -tieoff 5	Hierarchical tie-off connection	require_value -name top.dmass.DMA.P2[2] -value 1 require_value -name top.dmass.DMA.P2[1] -value 0 require_value -name top.dmass.DMA.P2[0] -value 1

Connection command	Type of command	Generated SGDC
add_hier_connection -source top::i0::i1::i2::P1[7:6] -dest top::i0::i4::P2[3:2] -through {top::i0::i5 top::i7 top::i8::i9}	Hierarchical pin-to-pin connection through specified through instances	require_path -from top.i0.i1.i2.P1[7] -to top.i0.i4.P2[3] require_path -from top.i0.i1.i2.P1[6] -to top.i0.i4.P2[2]
delete_hier_connection -source p -dest dsg_inst::dd_inst::p	Hierarchical connection deletion	illegal_path -from top.p -to top.dsg_inst.dd_inst.p
delete_connection -instance I1 -pin P1 -lsb 0 -msb 1 -instance dmass::DMA -pin P2 -lsb 0 -msb 1	Pin-to-pin connection deletion	illegal_path -from top.I1.P1[0] -to top.dmass.DMA.P2[0] illegal_path -from top.I1.P1[1] -to top.dmass.DMA.P2[1]
delete_connection -instance I1 -pin P1 -tieoff 5	Tie-off connection deletion	illegal_value -name top.I1.P1[2] -value 1 illegal_value -name top.I1.P1[1] -value 0 illegal_value -name top.I1.P1[1] -value 1

Example

This example shows the SGDC constraints that are generated for a sample Tcl file called test.tcl. To enable SGDC constraints generation, use the following Tcl command:

```
set_gensys_options -write_connectivity_check_sgdc_file true
```

TCL Script - test.tcl

```
new design top
add_port -name clk -direction IN
add_instance -name u1 -master morpheus8_core
add_connection -instance u1::controller0 -pin rdn -instance
u1::controller1 -pin add_in
add_connection -instance u1::controller0 -pin clk -port clk
add_connection -instance u1::controller0 -pin finish -tieoff 1'b1
add_connection -instance u1::controller0 -pin rdaddr -lsb 1 -msb 2
-tieoff 2'b10
add_hier_connection -source {top::u1::controller1::my_out[0:1]} -dest
{top::u1::controller0::my_sig[0:1]} -through top::u1::u2
delete_hier_connection -source clk -dest top::u1::controller0::clk
select_design morpheus8_core
delete_connection -instance controller0 -pin rdaddr -lsb 1 -msb 2 -tieoff
2'b10
select_design top
run GenerateVerilog
```

During RTL generation, GenSys generates a support file containing the SGDC constraints in the report directory. In this example, the -connectivity_check_sgdc_filename argument has not been specified in the set_gensys_option Tcl option. Therefore, by default, the generated SGDC file is called verify_connections.sgdc and it is located in the reports directory.

Generated SGDC Constraints File - verify_connections.sgdc

```
current_design top
require_path -from top.u1.controller1.add_in -to top.u1.controller0.rdn
require_value -name top.u1.controller0.finish -value 1
require_path -from top.u1.controller1.my_out[1] -to
top.u1.controller0.my_sig[1]
require_path -from top.u1.controller1.my_out[0] -to
top.u1.controller0.my_sig[0]
```

To verify these connections in the GenSys-generated RTL, use the generated SGDC file and the GenSys-generated RTL after assembly operations as an input for the SpyGlass Connectivity Verify product.

Limitations

This feature has the following limitations:

- Do not use this feature for splice connections and reverse connections.
- Do not use this feature for connections from a glue or generate block, or for connections inside a generate block.

- SGDC constraints are not be generated for restructuring operations.

6

Building SoCs

GenSys provides you the flexibility to build SoCs or designs by using the top-down, bottom-up, or the mix of these approaches. Using these approaches, you can either create designs either from scratch or modify existing designs (derivative designs).

Understanding the Top-Down Approach

In this approach, you build SoCs by starting with a top-level chip architectural specification and create black box definitions as needed for any top level IP.

You instantiate these IPs in a top level design, and make connections to establish the specification.

In this approach, you can divide work among design teams to complete design details in different parts of the design.

Understanding the Bottom-Up Approach

This is the traditional approach to build SoCs.

In this approach, you build a subsystem (design) by interconnecting components (IPs).

Once you build the subsystem, you can instantiate it in a design, just as if it was a component (IP) and continue with creating and updating connections in the design.

Building Designs from Scratch

This section describes the following tasks to build designs from scratch:

- Specifying a Default Save Format for the Design
- Creating a New Design
- Adding Ports on the Design
- Adding Instances in the Design
- Modifying Instance Parameters
- Adding Glue Logic
- Creating Connections
- Controlling Port and Net Names of Connections
- Exporting Interfaces and Ports on the Design
- Updating Master
- Updating the Design Hierarchy
- Saving the Design

- Viewing Reports
- Checking Design Health
- Generating the RTL

Specifying a Default Save Format for the Design

Use the set_default_save_format Tcl command to specify a default format in which the design should be saved.

Creating a New Design

You can create designs in any of the following ways:

- Using the GUI Menu Option to Create Designs
- Using the Canvas View to Create Designs
- Using Tcl to Create Designs
- Using MS Excel to Create Designs

Note:
By default, GenSys automatically infers the default extension as .xml. Do not use other extensions.

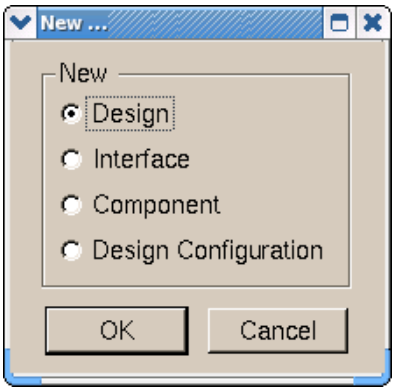
Using the GUI Menu Option to Create Designs

To create a design, perform the following steps:

1. Select the File->New menu option. Alternatively, click

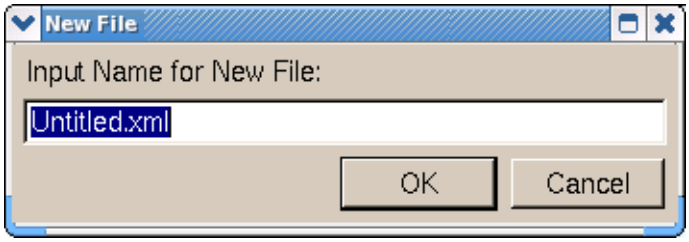
2. icon in the toolbar.

   The New dialog appears, as shown in the following figure:



3. Select the Design option in the above dialog and click the OK button.

   The New File dialog appears, as shown in the following figure:



4. Specify the name of the new design in the above dialog, and click the OK button.

Using the Canvas View to Create Designs

For details, see Creating New Objects.

Using Tcl to Create Designs

Use the new design <des-name> Tcl command to create a new design.

For details, refer to the *GenSys Tcl Commands Reference Guide*.

Using MS Excel to Create Designs

You can create a design by specifying its details in an MS Excel spreadsheet (XML SpreadSheet 2003), and then importing that sheet in GenSys.

The following figure shows the spreadsheets containing instance and interface details of a design:

	A	B	C	D	E	F	G	H	I
1	Component				Instance				
2	Vendor	Library	Name	Version	Name	Parameter Name	Parameter Value		
3	TI_AEC	TMS570	arm1156	1.0.0	arm1156_0				
4	TI_AEC	TMS570	arm1156	1.0.0	arm1156_1				
5	TI_AEC	TMS570	arm1156	1.0.0	arm1156_2				
6	TI_AEC	TMS570	FABRIC	1.0.0	FABRIC_0				
7	TI_AEC	TMS570	PLL	1.0.0	PLL				
8									
9									

Design | Instances | Interface | AdHoc | Exports | Tieoffs | Parameters | Attributes

Ready

	A	B	C	D	E	F	G	H
1	Source			Destination			Description	
2	Instance	Interface	Splice	Instance	Interface	Splice		
3	arm1156_0	AXI		FABRIC_0	AXI_M0		Connecting AXI Bus	
4	arm1156_1	AXI		FABRIC_0	AXI_M1		Connecting AXI Bus	
5	arm1156_2	AXI		FABRIC_0	AXI_M2		Connecting AXI Bus	
6								
7								

Design | Instances | Interface | AdHoc | Exports | Tieoffs | Parameters | Attributes

Ready

Once you fill in the details in the spreadsheet, import that spreadsheet in GenSys by using the following Tcl command:

```
import_data -file <filename.xml> -type excel
```

You can also create a design by using MS-Excel using the default templates provided at $GENSYS_HOME/auxi/Genesis/features/sdi/gallery/excel/design/sample_des.xml.

Adding Ports on the Design

You can ports on a design in any of the following ways:

- Using the Canvas View to Add Ports
- Using the Tabular View to Add Ports

- Using Tcl to Add Ports

Using the Canvas View to Add Ports

For details, see Adding Ports.

Using the Tabular View to Add Ports

To add ports on a design in the tabular view, perform the following steps:

1. Double-click on the design in the Library tab to load the design in GenSys.
2. Click on the Ports tab.
3. Click

4. button from the toolbar to add a new row.
5. Specify the port details in the newly added row.
6. Repeat step 3 to add more ports.

The following figure shows the ports information in the table in the Ports tab:

Info | Name | Ports | instances | Connections | Parameters | Views

	Name	direction	MSB	LSB	align	lsb_bit	Interfaces
0	HADDRhdma0	OUT	31	0	MSB	RIGHT	[31:0]:AHBLi...
1	HBURSThdm...	OUT	2	0	MSB	RIGHT	[2:0]:AHBLit...
2	HMASTLOCK...	OUT			MSB	RIGHT	AHBLite_hd...
3	HPROThdma0	OUT	3	0	MSB	RIGHT	[3:0]:AHBLit...
4	HRDATAhdm...	IN	31	0	MSB	RIGHT	[31:0]:AHBLi...
5	HREADYhdm...	IN			MSB	RIGHT	AHBLite_hd...

Using Tcl to Add Ports

Use the add_port Tcl command to add ports.

For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

Adding Instances in the Design

Add instances in a design in any of the following ways:

- Using the Canvas View to Add Instances
- Using the Tabular View to Add Instances
- Using Tcl to Add Instances

Using the Canvas View to Add Instances

To add instances in a design by using the canvas view, perform the following steps:

1. Double-click on a design in the Library Browser tab to load the design in the schematic.
2. Enter into the edit mode. For details, see Entering into the Edit Mode.
3. In the Library Browser tab, select the object whose instance you want to add in the currently loaded design.

   The object can be a design, component, or interface.
4. Drag the selected object in the schematic.

After performing the above steps, the instance of the selected object appears in the schematic and also under the hierarchy of the design in the Design Browser tab.

Modifying Instance Names

GenSys generates instance names automatically. To modify the instance name, perform the following steps:

1. Enter into the edit mode. For details, see Entering into the Edit Mode.
2. Select the instance.
3. Modify the instance name in the *Properties* tab of the *Property Editor* pane.

Using the Tabular View to Add Instances

To add instances in a design by using the tabular view, perform the following steps:

1. Double-click on a design in the Library Browser tab to load the design.
2. Click the Instances tab.
3. Click

4. button from the toolbar to add a new row.
5. Specify the instance details in the newly added row.
6. Repeat step 3 to add more rows.

The following figure shows the instances details in the *Instances* tab:

Info | Name | Ports | instances | Connections | Parameters | Views | Files

	name	master_name	master_version	master_vendor	master_libr
0	u_Lite2AhbX...	Lite2AhbXf	1.0.0	ATRENTA	AHBLIB
1	u_ahb_NFC	nflash_top	1.0.0	ATRENTA	AHBLIB
2	u_stmcdecode	stmcdecode	1.0	ATRENTA	lib1
3	u_Lite2AhbX...	Lite2AhbXf	1.0.0	ATRENTA	AHBLIB
4	u_ST_MemCt...	ST_MemCtrl...	1.0.0	ATRENTA	AHBLIB
5	u_AHB_HDM...	AHB_HDMA	1.0.0	ATRENTA	AHBLIB
6	u_AHB_HDM...	AHB_HDMA	1.0.0	ATRENTA	AHBLIB
7	u_AHB_MAC_0	AHB_MAC	1.0.0	ATRENTA	AHBLIB
8	u_AHB_MAC_1	AHB_MAC	1.0.0	ATRENTA	AHBLIB
9	u_otg_ctrl_wr...	otg_ctrl_wra...	1.0.0	ATRENTA	AHBLIB
10	u_CI12323cn...	Leopard6_CI...	1.0	ATRENTA	lib1

Using Tcl to Add Instances

Use the add_instance command to add instances in a design.

For details on this command, refer to the *Tcl Commands Reference Guide*.

Modifying Instance Parameters

Once you instantiate a component on a design, you can modify the parameters of that instance in any of the following ways:

- Using the Right-Click Option in the Design Browser tab
- Using the Property Editor Pane in the Canvas View
- Using Tcl

Using the Right-Click Option in the Design Browser tab

To modify instance parameter, right-click on an instance in the Design Browser tab and select the Reconfigure Instance option from the shortcut menu.

Using the Property Editor Pane in the Canvas View

For details, see Using the Canvas View to Assign Parameters.

Using Tcl

Use the set_parameter Tcl command to modify instance parameter.

For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

Adding Glue Logic

For details, see Adding Glue Logic in a Design.

Creating Connections

For details, see Creating Connections.

Controlling Port and Net Names of Connections

GenSys provides you the flexibility to control the names of nets and ports created while generating connections.

Controlling Net Names

For details, see Controlling Net Names while Creating Connections.

Controlling Port Names

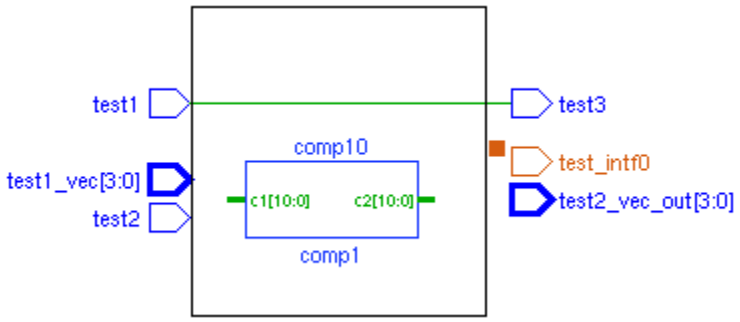
You can control the names of GenSys generated ports by using the set_default_port_name Tcl command.

For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

Exporting Interfaces and Ports on the Design

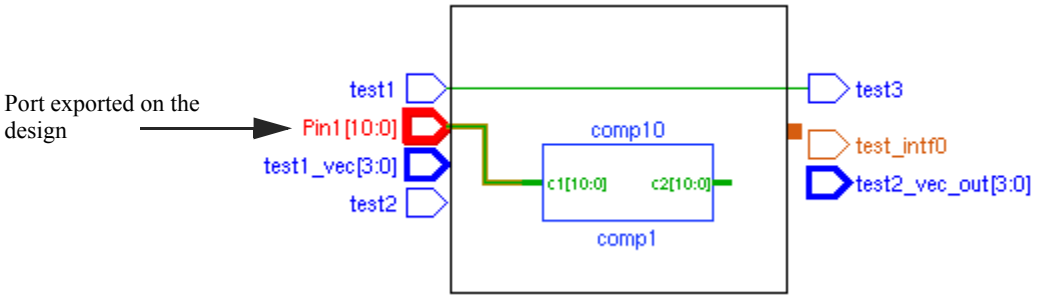
Exporting an interface or a port means making the interface of an IP or the port of an instance visible at the top-level of the design.

For example, consider the comp10 instance in the Des design, as shown in the following figure:



Des

In the above scenario, when you export the C1 port of the comp10 instance on the Des design, the design containing the exported port appears, as shown in the following figure:



Des

You can export interfaces and ports on a design in any of the following ways:

- Using the Canvas View to Export Pins and Interfaces
- Using the Connection Builder Dialog to Export Interfaces and Ports
- Using Tcl to Export Interfaces and Ports

Using the Canvas View to Export Pins and Interfaces

For details, see Exporting Pins an Interfaces to the Design Boundary.

Using the Connection Builder Dialog to Export Interfaces and Ports

For details, see Exporting an Instance Pin/ Interface Port.

Using Tcl to Export Interfaces and Ports

To export an interface on the design boundary, use the export_interafce Tcl command.

To export the port of an instance on the design boundary, use the export_pin Tcl command.

For details on these commands, refer to the *GenSys Tcl Commands Reference Guide*.

Updating Master

While building SoCs, the IP versions may get updated multiple times.

To ensure that you use the latest version of the IP, you need to upgrade the older version of the IP to its new version in any of the following ways:

- Using GUI to Update to the New Version of the IP
- Using Tcl to Update to the New Version of the IP

Using GUI to Update to the New Version of the IP

Right-click on the IP in the Design Browser tab, and select the *Update Masters* option from the shortcut menu.

For details, see Update Masters.

Using Tcl to Update to the New Version of the IP

Use the update_master Tcl command to upgrade the IP to its latest version.

For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

Updating the Design Hierarchy

See Tasks in the RTL Restructuring Flow.

Saving the Design

Save the design by using the *File -> Save* menu option.

Alternatively, use the save or export_data Tcl command. For details on these commands, refer to *GenSys Tcl Commands Reference Guide*.

Viewing Reports

You can view the following reports to check the status of the newly created design:

- Design Summary
- Interface Summary
- Connection Summary
- Suggested Interfaces
- Suggested Interfaces

Checking Design Health

For details, see Checking the Design Health.

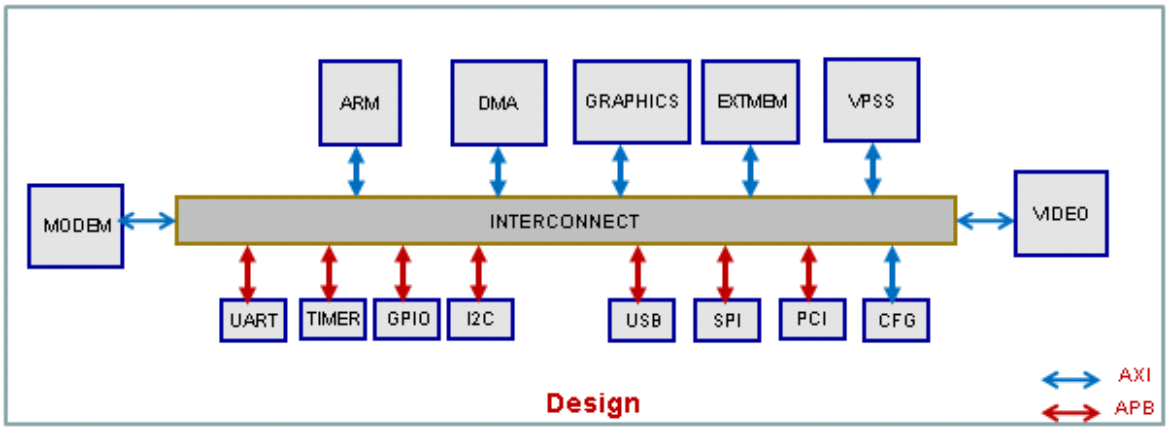
Generating the RTL

For details, see Generating RTL.

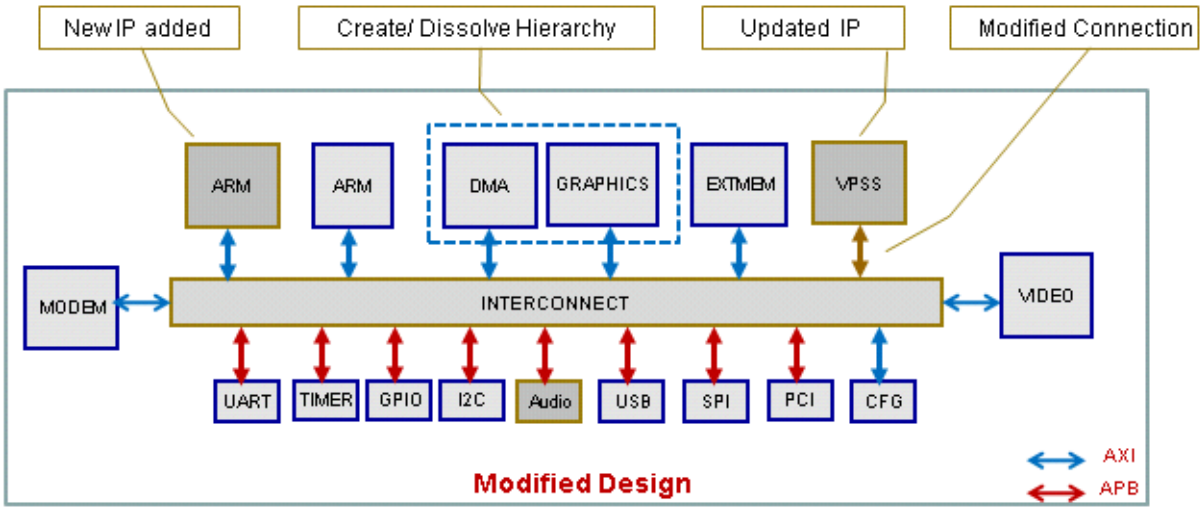
Building Derivative Designs

Derivative designs are the existing legacy designs on which the designer performs changes based on specific requirements.

For example, consider the following design structure:



Once you load the above design in GenSys, make modifications as per the new requirements, as shown in the following figure:



Loading Existing Designs

You can load existing designs in GenSys in any of the following ways:

- Loading the Existing XML of the Design
- Loading the Design from the Library Browser
- Loading the Design RTL

- Importing MS Excel Containing Design Data

Loading the Existing XML of the Design

The XML of the design can be in the IP-XACT or the native GenSys MRS format.

Derivative design can be rapidly created if the existing design is already in any of these formats.

To load the XML, perform any of the following actions:

- Select the *File -> Open* menu option.
- Use the load Tcl command. For details, refer to the *GenSys Tcl Commands Reference Guide*.

Loading the Design from the Library Browser

Use the gload Tcl command to load the design from the Library Browser.

For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

Loading the Design RTL

You can load the RTL of a design in any of the following ways:

- Through GUI. For details, see Importing RTL Through GUI.
- Through Tcl. For details, see Importing RTL by Using Tcl Command

Importing MS Excel Containing Design Data

For details, see Using MS Excel to Create Designs.

Abstracting Interface Connections

For any non packaged IP instantiated in an RTL design, perform interface mapping by using the infer_interfaces Tcl command.

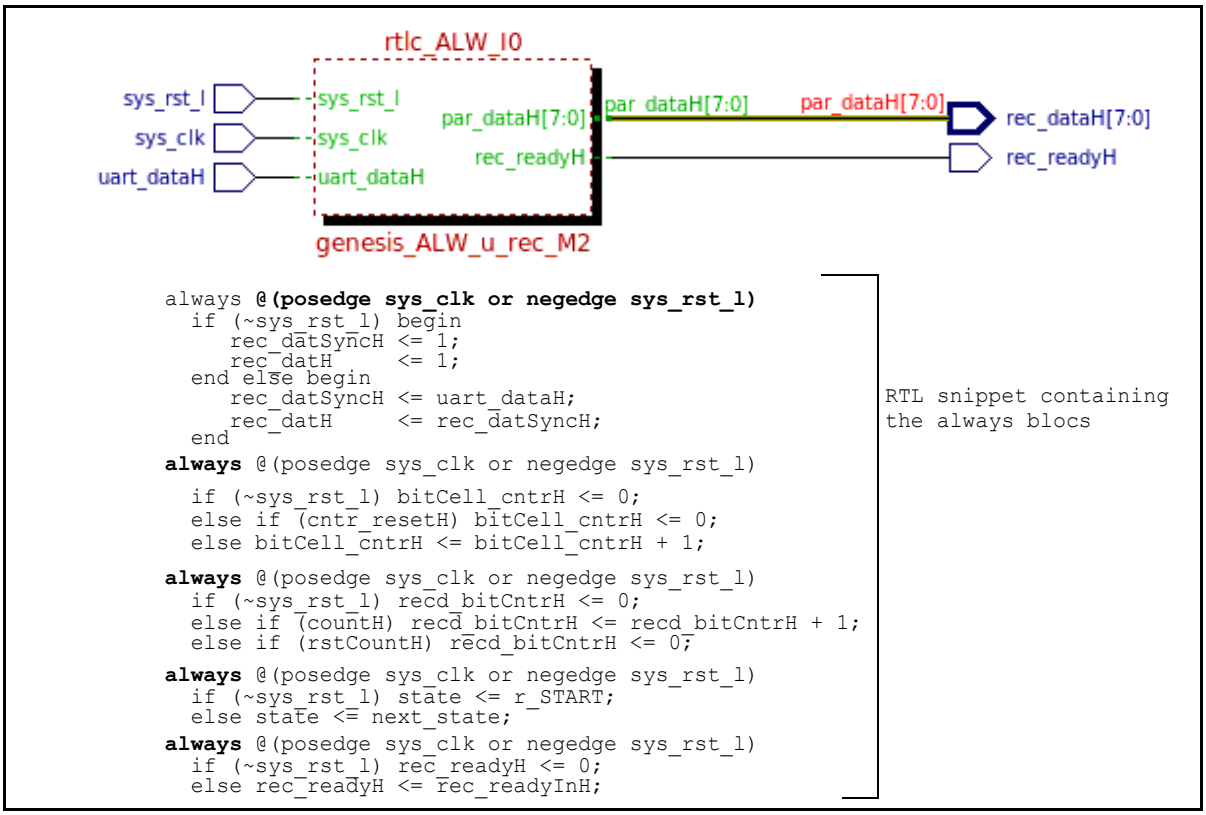
For details on interface mapping, see Mapping Interfaces.

Handling Glue Logic During RTL Import

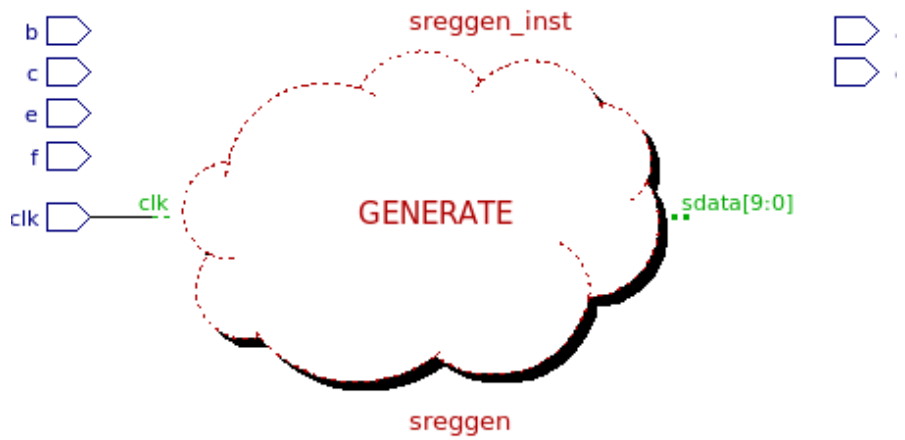
For details, see Inferring Glue Logic During RTL Import.

Solidifying Glue Instances and Generate Blocks

A design may contain multiple glue logic in the form of *assign* and *always*/*process* blocks. All such blocks are clubbed under one virtual hierarchy that appears in the form of a dotted box, as shown in the following figure:



Similarly, if a design contains a *generate* statement, that statement appears as a separate dotted cloud. If there are multiple *generate* statements, multiple clouds appear, one for each statement. This is shown in the following figure:




```
generate
 for(i=0; i<SIZE; i=i+1) begin:sreggen
   sreg s(clk, sdata[i], sdata[i+1]);
 end
 endgenerate
```

By default, you cannot move a glue logic or a *generate* block across hierarchies. In addition, the design containing a *generate* block cannot be ungrouped.

To move a glue logic or a *generate* block across hierarchies or to enable the ungrouping of a design containing a *generate* block, you first need to make the glue logic or *generate* block an independent instance. This process is known as solidification because the glue logic or *generate* block no more appears in a dotted hierarchy. Rather, it appears as a solid box similar to other instances that can be moved across hierarchies.

You can perform solidification through Tcl by using the group_glue Tcl command or through GUI. For information on solidifying through GUI, see Solidifying Individual Glue Logic from a Glue Group.

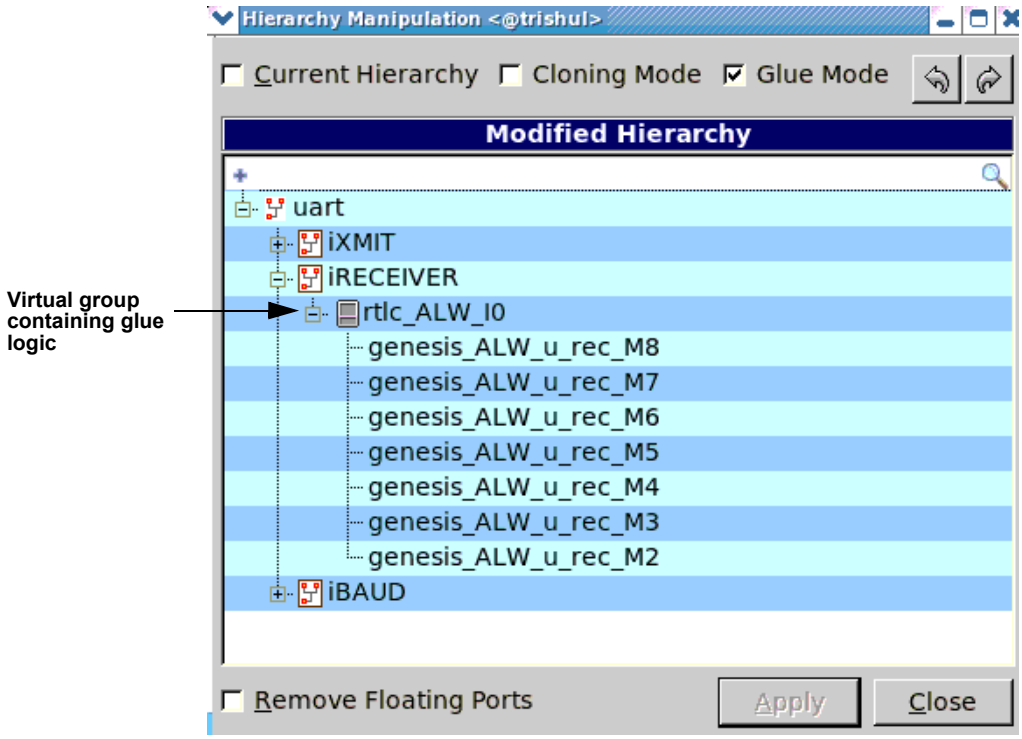
You can also solidify the entire virtual hierarchy (containing multiple glue logic) so that it appears as one instance that can be moved across hierarchies. For details, see Solidifying a Glue Group.

GenSys generates separate files for the solidified glue instances. For details, see Files Generated for Solidified Glue Logic.

Solidifying Individual Glue Logic from a Glue Group

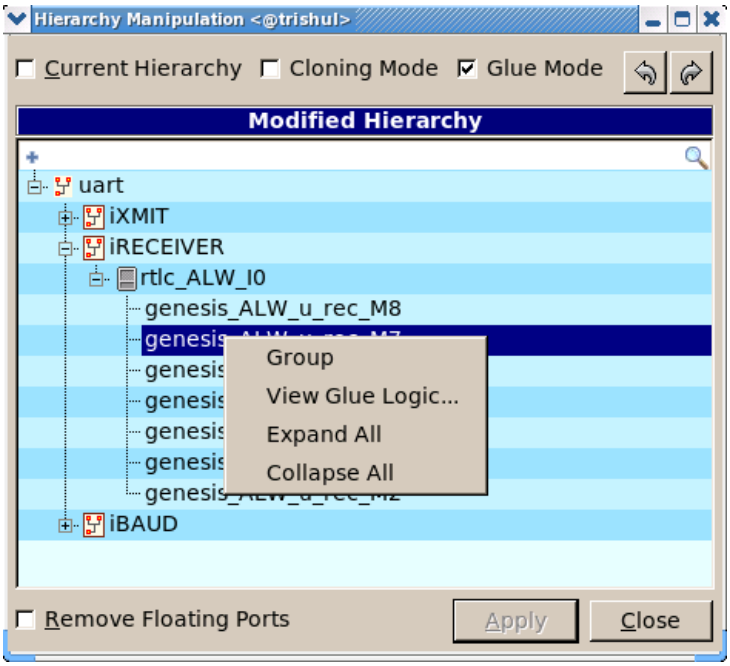
To perform glue solidification, perform the following steps:

1. Set the -enable_group_glue argument of the set_gensys_options Tcl command to TRUE.
2. Load the RTL in which you want to solidify glue logic. For details on loading RTL, see Performing RTL Import.
3. Open the *Hierarchy Manipulation* dialog.
4. Select the *Glue Mode* check box in this dialog.
5. Expand the virtual group containing one or more glue logic. The following figure shows the expanded glue logic list of the virtual group:

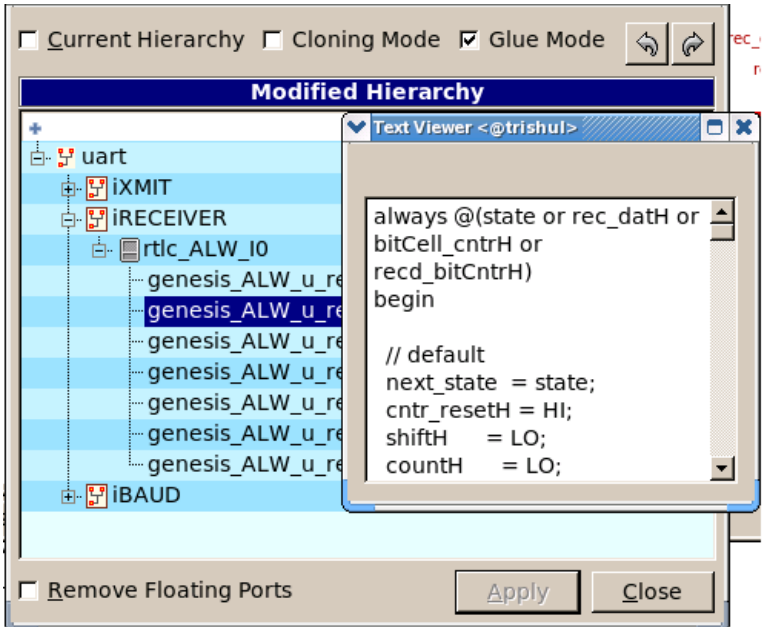


6. View the RTL of the glue logic that you want to solidify.

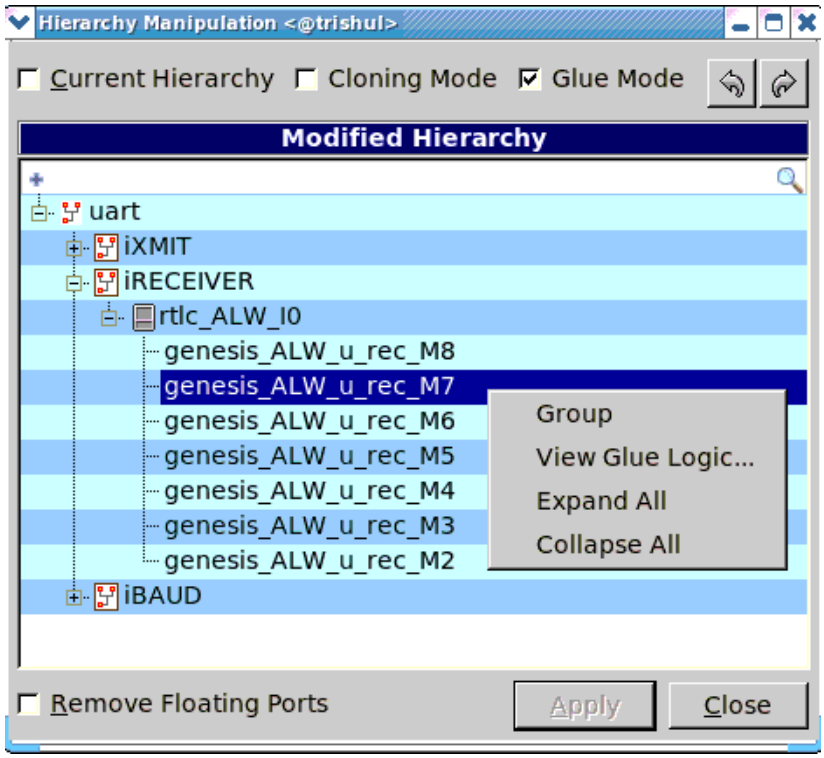
   To view the RTL, right-click on the glue logic and select the *View Glue Logic* option from the shortcut menu, as shown in the following figure:



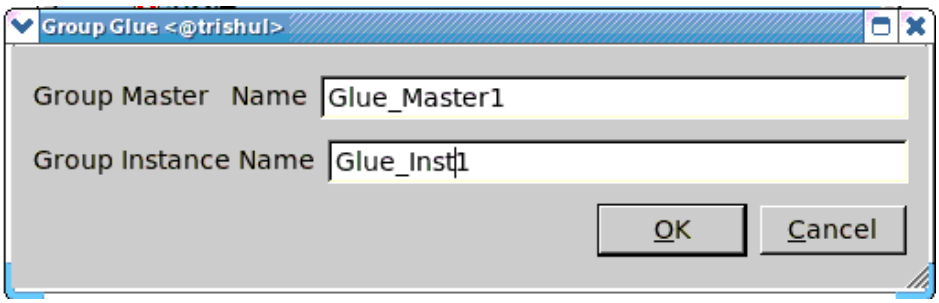
In the above dialog, when you select the *View Glue Logic* option, the RTL of that glue logic appears in a separate window, as shown in the following figure:



7. After viewing the RTL of the glue logic, select the glue logic that you want to solidify, right-click on the selected logic, and select the *Group* option from the shortcut menu, as shown in the following figure:



After performing this step, the following *Group Glue* dialog appears:



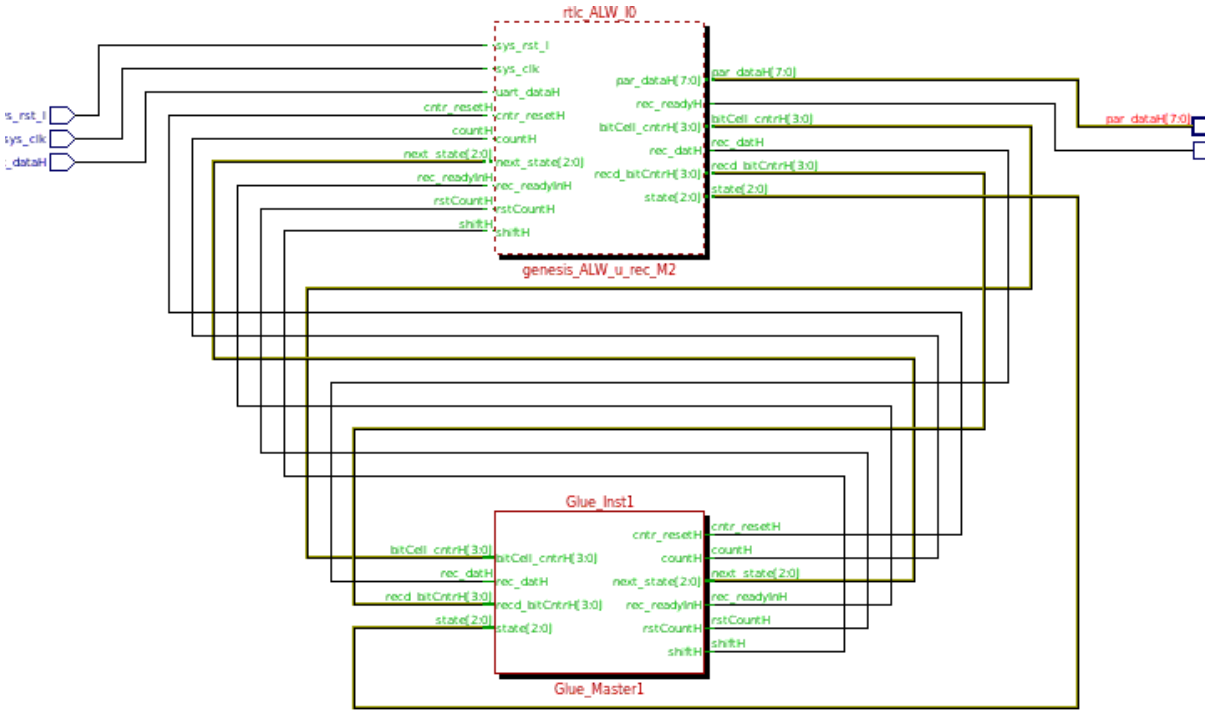
8. In the *Group Glue* dialog, specify the name of the instance in which the glue logic should be stored and the name of the master of that instance so that the master name is saved in the generated RTL.

9. Click the *OK* button to close the *Group Glue* dialog.

   After performing the above steps, the selected *Genesis_ALW_u_rec_M7* glue logic shown in the figure is moved out from the virtual group *rtlc_ALW_10*.

10. Click the *Apply* button in the *Hierarchy Manipulation* dialog to save the changes.

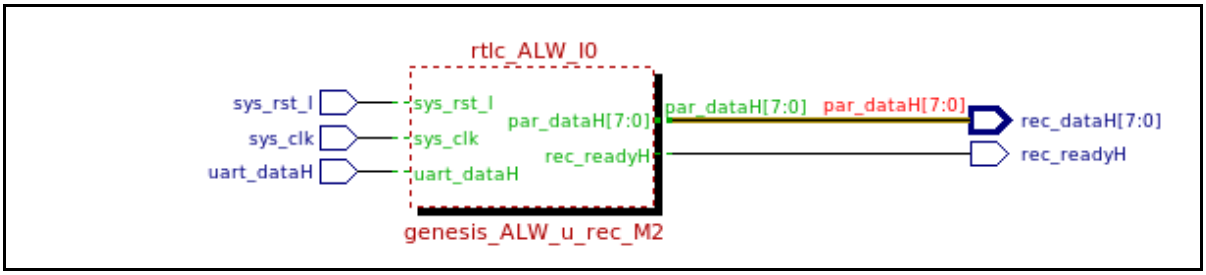
After performing the above steps, the selected glue logic now appears as a separate instance, *Glue_Inst1*, as shown in the following figure:



GenSys maintains all the connectivity of the earlier glue logic (now a glue instance).

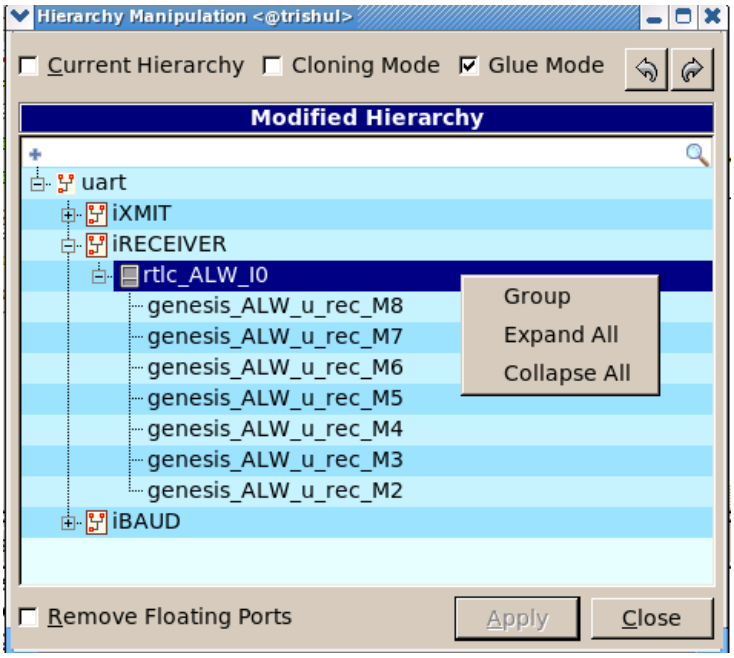
Solidifying a Glue Group

Consider the following glue group:

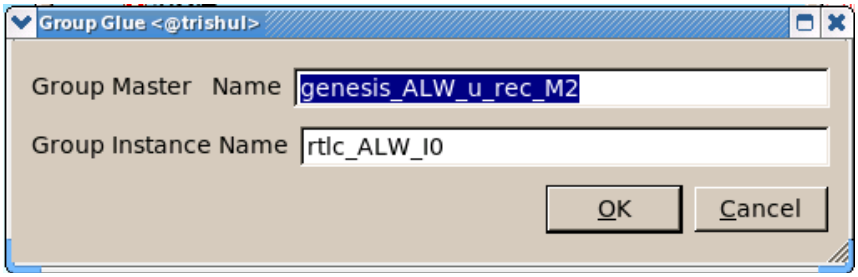


To solidify the above glue group, perform the following steps:

1. Set the -enable_group_glue argument of the set_gensys_options Tcl command to TRUE.
2. Load the RTL in which you want to solidify glue logic. For details on loading RTL, see Performing RTL Import.
3. Open the *Hierarchy Manipulation* dialog.
4. Select the *Glue Mode* check box in this dialog.
5. Right-click on the glue group to be solidified, and select the *Group* option from the shortcut menu, as shown in the following figure:

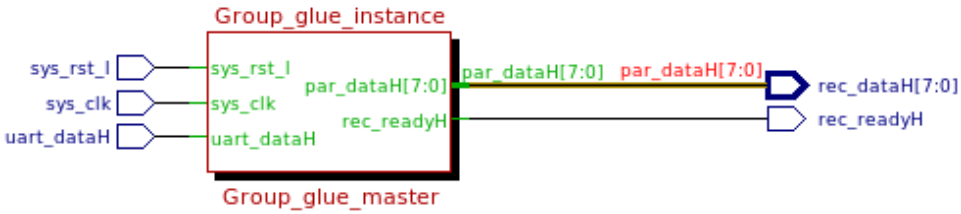


After performing this step, the *Group Glue* dialog appears, as shown in the following figure:



6. In the *Group Glue* dialog, specify the master name (say *Group_glue_master*) and instance name (say *Group_glue_inst*) to be created for the glue group.
7. Click the OK button to close the *Group Glue* dialog.
8. Click the *Apply* button in the *Hierarchy Manipulation* dialog to save the changes.

After performing the above steps, the glue group appearing in the figure becomes the *Group_glue_inst* instance, as shown below:



You can now move the above instance, containing multiple glue logic, across design hierarchies.

Files Generated for Solidified Glue Logic

When you generate RTL, GenSys creates separate Verilog/VHDL files for each solidified glue logic and/or glue logic group.

For example, for the solidified glue logic group shown in the figure, GenSys generates the *Group_glue_master.v* file.

Updating Instance Details

You can update instance details in a design in any of the following ways:

- Adding Instances in the Design
- Assigning Parameters
- Deleting instances through any of the following ways:
  - GUI (See Deleting Objects)
  - delete_instance Tcl command (See *GenSys Tcl Commands Reference Guide*)

Managing Connections

For details, see:

- Creating Connections
- Controlling Net Names while Creating Connections
- Managing Overridden Connections
- Generating Tie-off Connections

Customizing Ports and Net Name

For details, see Controlling Port and Net Names of Connections.

Updating the Master of an IP

For details, see Updating Master.

Performing RTL Health Check

For details, see Checking the Design Health.

Performing Hierarchy Manipulations

For details, see Tasks in the RTL Restructuring Flow.

Comparing Different Versions of an Object

In the current competitive scenario, SoC designers heavily depend upon IP reuse in order to shorten time-to-market. The IPs being reused are generally customized as per requirement.

However, as the SoC development proceeds, the following cases may occur:

- Multiple versions of an IP may exist simultaneously as different designers may work on different parts of an IP.
- The same IP may be revised multiple times giving rise to multiple incremental versions.

In any of the above cases, you may need to view the difference between two versions of an IP. In such cases, you can use the Difference Merging Tool.

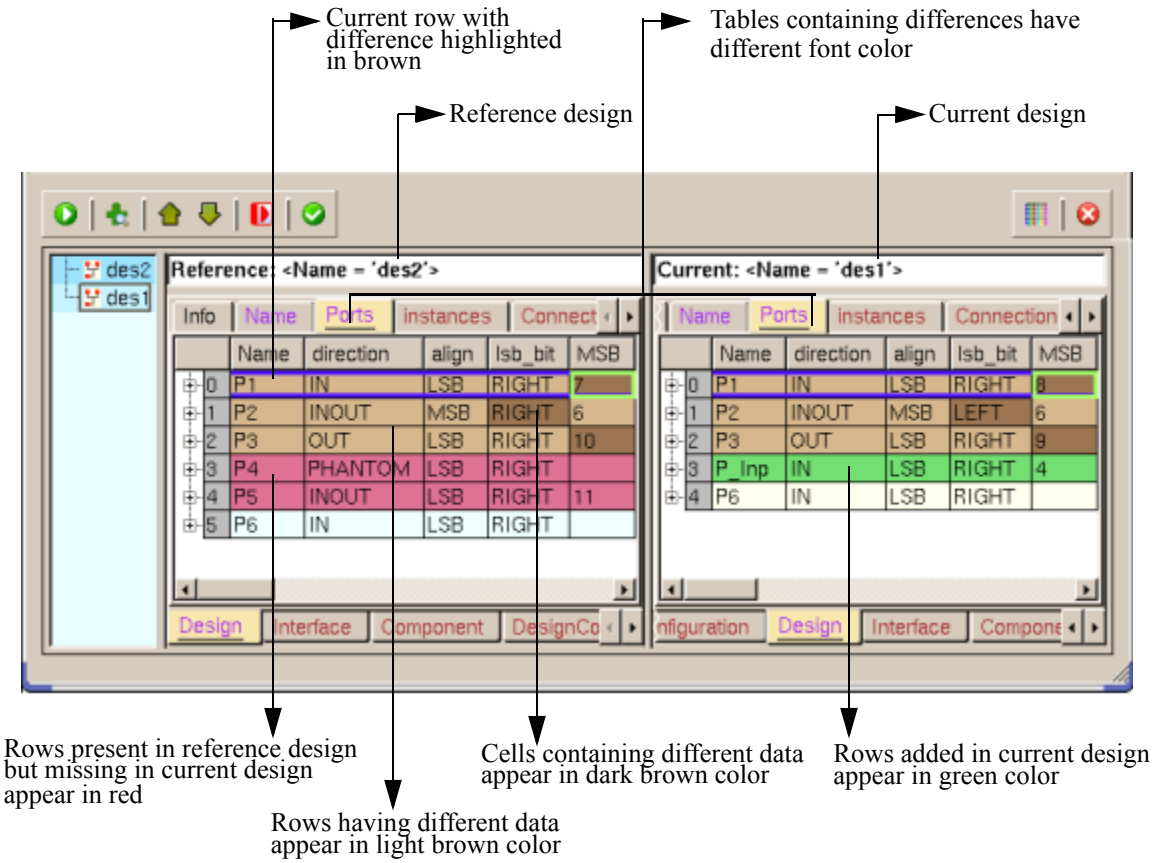
Introducing the Difference Merging Tool

The Difference Merging Tool is a feature in GenSys that enables you to compare or view differences between two versions of an object, such as a design, a component, a design configuration, or an interface. In addition, you can also merge the differences by using this feature.

The objects being compared are:

- Reference object: An object with which comparison needs to be made.
- Current object: An object that needs to be compared.

The following figure shows the difference between two versions of a design in the Difference Merging Tool:



In the above window, the reference design and current design are displayed simultaneously to enable comparison.

Different coloring conventions are used to highlight the differences. To set the colors as per your choice, click Show Color Legend button (

). When you click this button, the Legend dialog appears in which you can set the required colors.

Invoking the Difference Merging Tool

To invoke the Difference Merging Tool, perform the following steps:

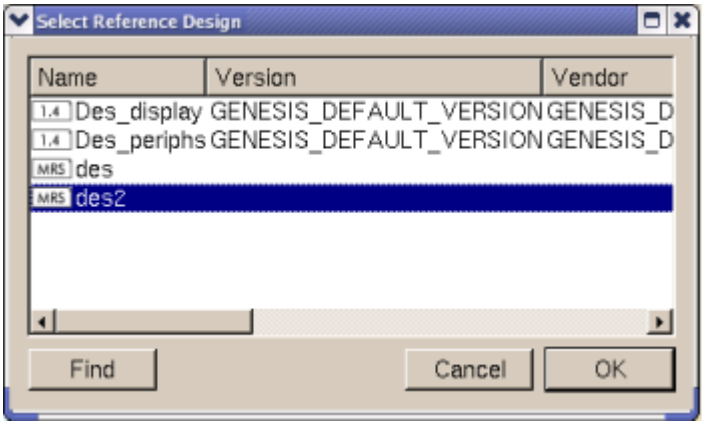
1. Double-click the object to be compared in Library Browser Tab.

This step loads the object to be compared in Design Browser Tab.

2. Click the Difference Merging Tool icon (

3.) in the toolbar. Alternatively, select Tools -> Difference Merging Tool menu option.

   This step displays the Select Reference Design dialog, as shown in the following figure:



4. Select a reference design in the Select Reference Design dialog.
5. Click the OK button to close the Select Reference Design dialog.

   This step invokes the Difference Merging Tool window.
6. Click

7. in the Difference Merging Tool window to view the differences.

Using the Difference Merging Tool

You can perform the following actions in the Difference Merging Tool window:

- Navigate across different rows containing changes by using the navigation buttons (

-).

- Sign-off the difference of a cell by clicking

- .

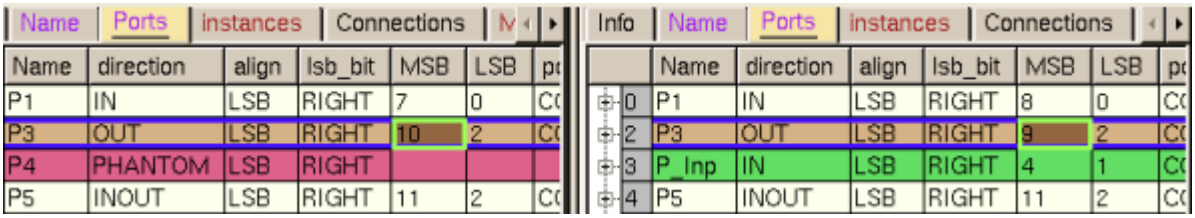
  Selecting this button removes the color indicating any difference in the selected cell. This way, you can focus only on those cells for which you want to view differences.

- Merge differences of the reference object in the current object by clicking

- .

  When you click this button, the selected row/cell of the reference object gets copied in the corresponding row/cell of the current object.

  For example, consider the following figure:



Name | Ports | instances | Connections

Name	direction	align	lsb_bit	MSB	LSB	p
P1	IN	LSB	RIGHT	7	0	C
P3	OUT	LSB	RIGHT	10	2	C
P4	PHANTOM	LSB	RIGHT			
P5	INOUT	LSB	RIGHT	11	2	C

Info | Name | Ports | instances | Connections

	Name	direction	align	lsb_bit	MSB	LSB	p
0	P1	IN	LSB	RIGHT	8	0	C
2	P3	OUT	LSB	RIGHT	9	2	C
3	P_Inp	IN	LSB	RIGHT	4	1	C
4	P5	INOUT	LSB	RIGHT	11	2	C

  In the above case:

  - The value (10) in the MSB column of the P3 port in reference design overwrites the value (9) in the MSB column of the P3 port in current design.
  - If you move to the row highlighted in pink and click

  - , that row gets copied in corresponding table in the Ports table of the current design.

- View differences on RTL health check tables of reference and current objects.

  To view this difference, click

  and then click

.

- Close this window by clicking

- .When you click this button, the main GenSys window appears.

Advantages of Difference Merging Tool

The Difference Merging Tool provides the following advantages:

- You can compare differences between two versions of the same object visually.
- You can invoke this tool any time during design development and compare the changes.
- You can invoke this tool directly from main GenSys window. Once you compare the changes, you can close this tool to automatically return to the main GenSys window.

  This way, you do not need to change you current working platform or switch from the current work to compare changes.

You can merge the data of reference object in the current object by using this tool.

Uniquifying Instances

Uniquifying instances is a process in which you modify the selected instances of a subsystem or a component.

All instances of a subsystem or a component are bound to a common master. If you update the master, all the instances get updated.

However, in the uniquification process, you create a new master of the selected instance and edit that master to replicate those changes in the selected instance. This way, all the other instances of the original master remain unchanged.

Using GUI to Uniquify Instances

You can uniquify instances in any of the following ways:

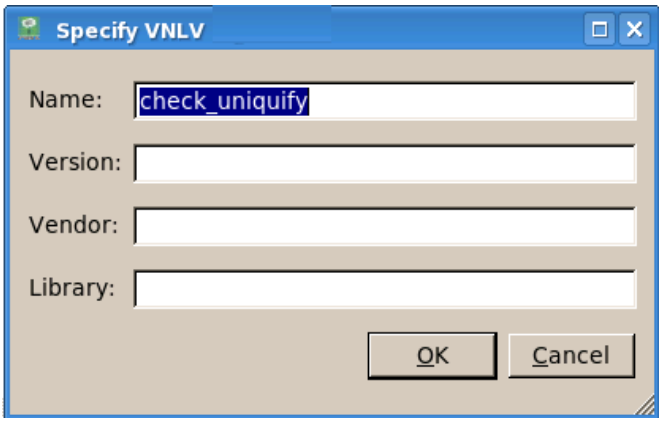
-
-

Uniquifying through Design Browser

To uniquify an instance through Design Browser Tab, perform the following steps:

1. Right-click on the instance in the Design Browser Tab or in the schematic and select the Uniquify Instance option from the shortcut menu.

   The Specify VNLV dialog appears, as shown in the following figure:



2. In the above dialog, specify the name of the new master for the selected instance and specify the Version, Vendor, and Library name of the new master.
3. Click the OK button.

After performing the above steps, GenSys creates a new master of the name specified in the above dialog and applies the specified VLV details of the new master to the selected instance.

Uniquify through Hierarchy Manipulation Widget

In the Hierarchy Manipulation widget, right-click on an instance and select the Uniquify option from the shortcut menu.

The dialog shown in the figure appears. Specify details and click the OK button.

Using Tcl to Uniquify Instances

Use the uniquify_instance Tcl command to uniquify instances.

For details on this command, refer to *GenSys Tcl Commands Reference Guide*.

Configuring Macros

You design may contain global macros affecting the IPs in a design. You can configure such macros during RTL import and/or during design assembly so that you can use the corresponding configuration of IPs based on the macros configured.

Configuring Macros During RTL Import

During RTL import, you can configure IPs by setting global-level macros used by the IPs. This way, you can import a particular configuration of IPs based on macro values.

To configure macros during RTL import, perform the following steps:

1. Select the Open Dialog to Mark Stop Module(s) & Macros check box in the Select RTL/ InterfaceLib Files dialog.

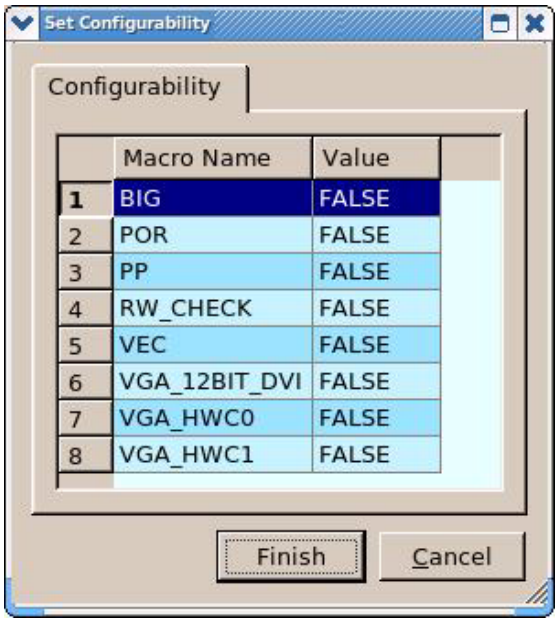
   See Importing RTL Through GUI.

2. Close the Select RTL/InterfaceLib Files dialog.

   The Specify Stop Modules & Macros dialog appears as shown in Figure 4-2.

3. Click the Configure Macros button in this dialog.

   The Set Configurability dialog appears displaying a list of global macros in the loaded RTL. See the following figure:



4. In the above dialog, set the required macros to TRUE so that the corresponding configuration of the IPs is loaded in GenSys.
5. Click the Finish button.

Configuring Macros During Design Assembly

During design assembly, you can instantiate IPs and configure their macros. For details, see Configuring Macros Through Schematic or Configuring Macros Through Tools -> Set Configurability/Visibility Menu

When you configure macros of IPs, GenSys enters into the synthesis mode in which it selectively imports such IPs from RTL with the configured macro values.

Once you configure macros and generate RTL, GenSys creates the defines.v file that contains details of the macros that were set to TRUE during RTL generation. This file name also appears as an entry in the sources.f file.

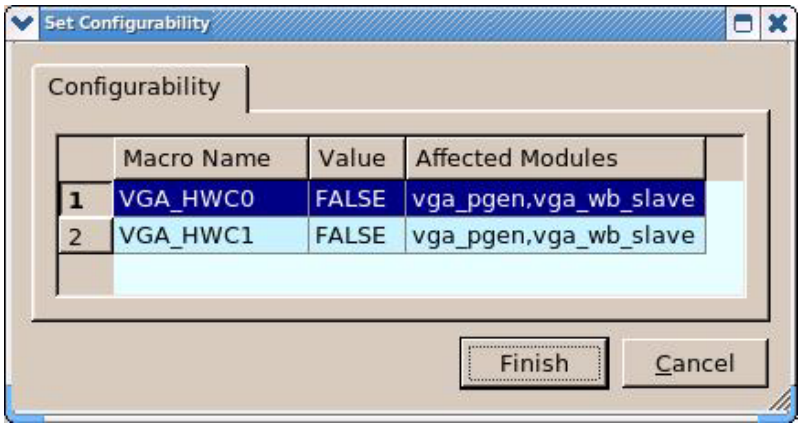
You can use the defines.v file to compare the new RTL with configured macros and the original RTL.

Configuring Macros Through Schematic

To configure macros through schematic, perform the following steps:

1. Enter in the edit mode. For details, see Entering into the Edit Mode.
2. Select the instances for which you want to configure macros by keeping the <*Shift*> keyboard key pressed.
3. Right-click on the instances and select the Set Configurability option from the shortcut menu.

   The Set Configurability dialog appears, as shown in the following figure:



The above dialog shows the modules affected by setting the listed macros.

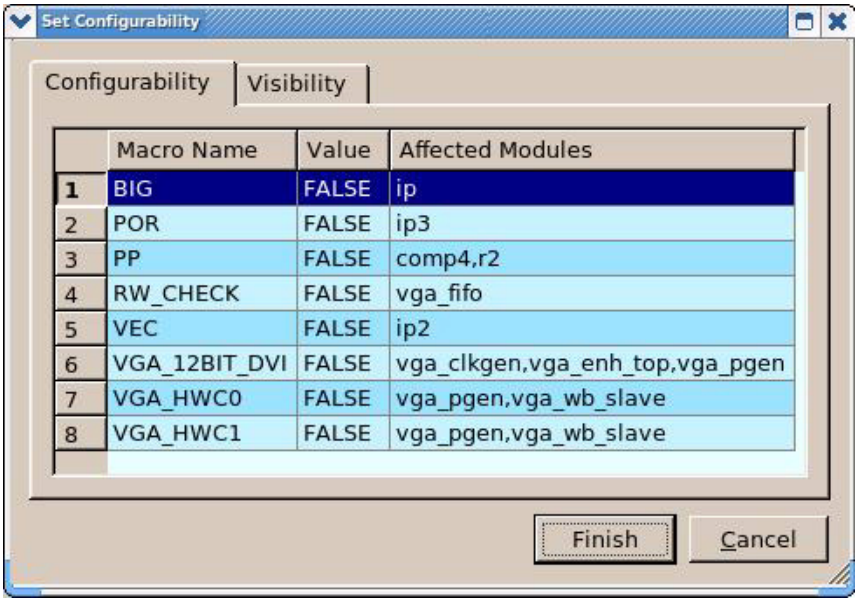
4. In the above dialog, change the value of the required macros to TRUE or FALSE.
5. Click the Finish button.

Configuring Macros Through Tools -> Set Configurability/Visibility Menu

To configure macros through the Tools -> Set Configurability/Visibility menu option, perform the following steps:

1. Enter in the edit mode. For details, see Entering into the Edit Mode.
2. Select the Tools -> Set Configurability/Visibility menu option.

   The Set Configurability dialog appears, as shown in the following figure:



	Macro Name	Value	Affected Modules
1	BIG	FALSE	ip
2	POR	FALSE	ip3
3	PP	FALSE	comp4,r2
4	RW_CHECK	FALSE	vga_fifo
5	VEC	FALSE	ip2
6	VGA_12BIT_DVI	FALSE	vga_clkgen,vga_enh_top,vga_pgen
7	VGA_HWC0	FALSE	vga_pgen,vga_wb_slave
8	VGA_HWC1	FALSE	vga_pgen,vga_wb_slave

3. Under the Configurability tab of the above dialog, change the value of the required macros to TRUE or FALSE.
4. Click the Finish button.

Controlling Visibility

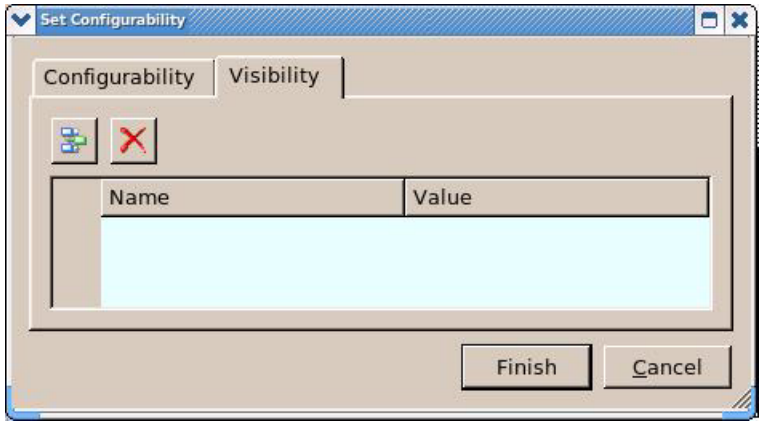
You can specify a visibility condition for ports, instances, and parameters such that based on the evaluated value of that condition (true or false), the ports, instances, and parameters are visible or hidden in the generated RTL.

To control visibility, perform the following steps:

1. Enter in the edit mode. For details, see Entering into the Edit Mode.
2. Select the Tools -> Set Configurability/Visibility menu option.

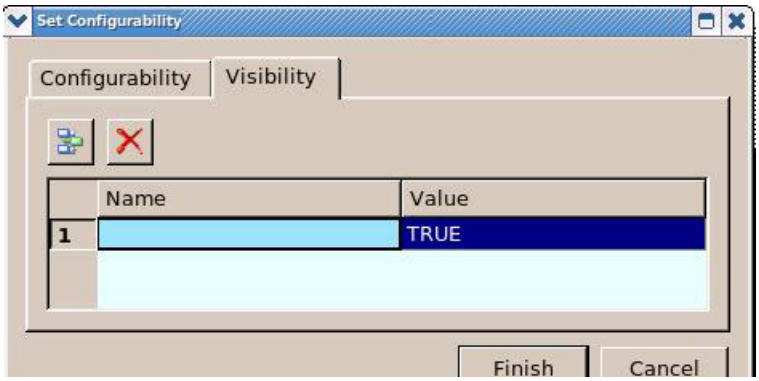
   The Set Configurability dialog appears.
3. Click the Visibility tab in the Set Configurability dialog.

   The following page appears in the Set Configurability dialog:



4. Click

5. to add a row in the above dialog.



6. Specify the name of the visibility condition and its value (TRUE or FALSE) in the above dialog:

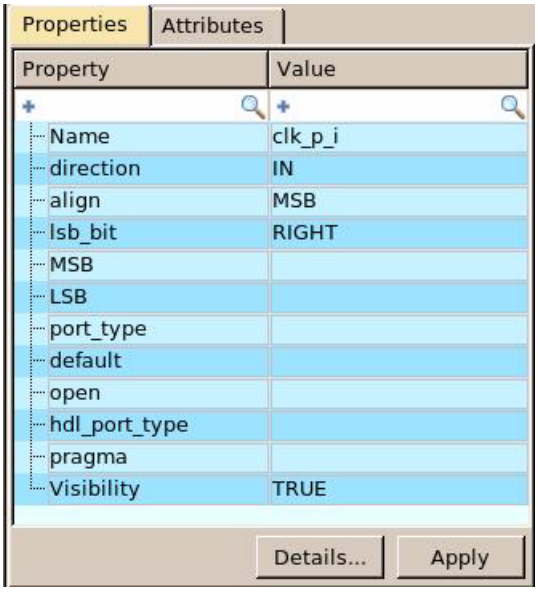
   In the above dialog, the visibility condition C1 is added with the value FALSE. Any object on which this visibility condition is applied will not appear in the generated RTL.

   Alternatively, use the add_gensys_define Tcl command to add a visibility condition. Use the set_gensys_define Tcl command to change the value (TRUE or FALSE) of the previously added visibility condition.

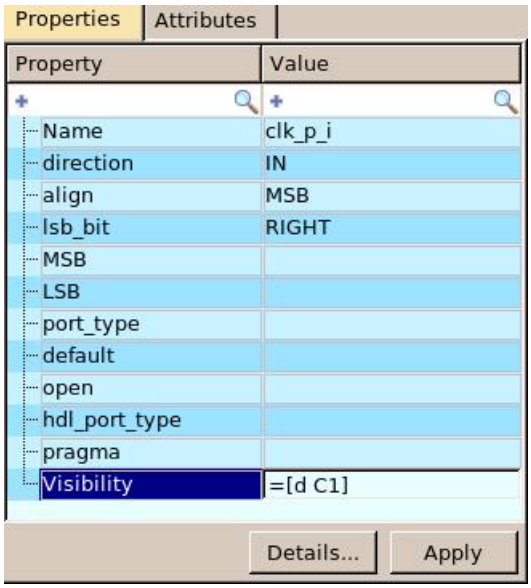
7. To delete a visibility condition, select that row and click

8. . Alternatively, use the remove_gensys_defines Tcl command.
9. Click the Finish button.
10. Open the Property Editor pane (View -> Views -> Property Editor).
11. Select an object (port, instance, or parameter) in the schematic.

    The properties of the selected object appear in the Property Editor pane. For example, the following figure shows the properties of the selected port clk_p_i:



12. In the above pane, specify the value =[d C1] in the Visibility field, as shown in the following figure:



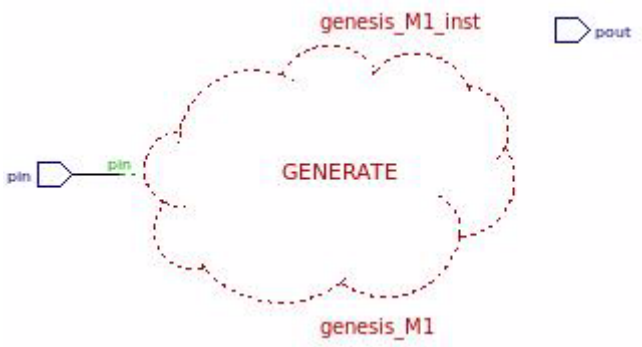
Alternatively, specify the expression in the -rtl_visibility argument of the add_port, add_instance, or add_parameter Tcl command.

13. Click the Apply button.

After performing the above steps, when you generate the RTL, the clk_p_i port does not appear in the generated RTL. This is because the evaluated value of the visibility condition (C1) based on the expression =[d C1]) is FALSE.

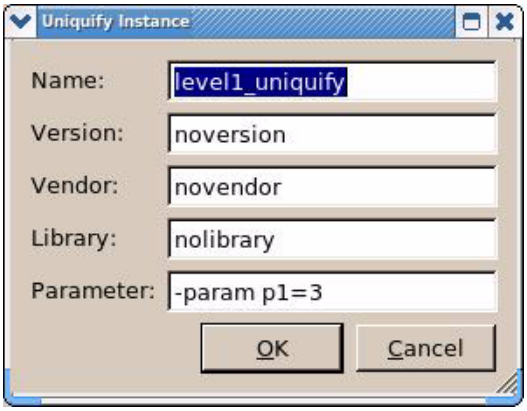
Working with Generate Blocks

GenSys imports the generate blocks in RTL as a pseudo hierarchy. Such blocks appear as cloud shaped instances in the schematic, as shown in the following figure:



- If you want to work on any of the instances present inside the generate block, you should first dissolve the pseudo hierarchy by using the Hierarchy Manipulation dialog. For details, see Dissolving a Hierarchy.
- If there are multiple instances of a master having the generate statements and you want to work on a particular instance, you must first uniquify that instance and then work on the uniquified instance. For details, see Uniquifying Instances.

  In the dialog to specify uniquified instance, GenSys provides an additional field to change the parameter values for such instances. See the following figure:



The Parameter field is not available for other instances.

- If you want to override the parameter value of the instance whose master has the generate statement, you must first uniquify that instance and then modify the parameter value of that instance by using the dialog shown in the figure.

  Note that you cannot modify the VLV of the instance in this case by using this dialog.

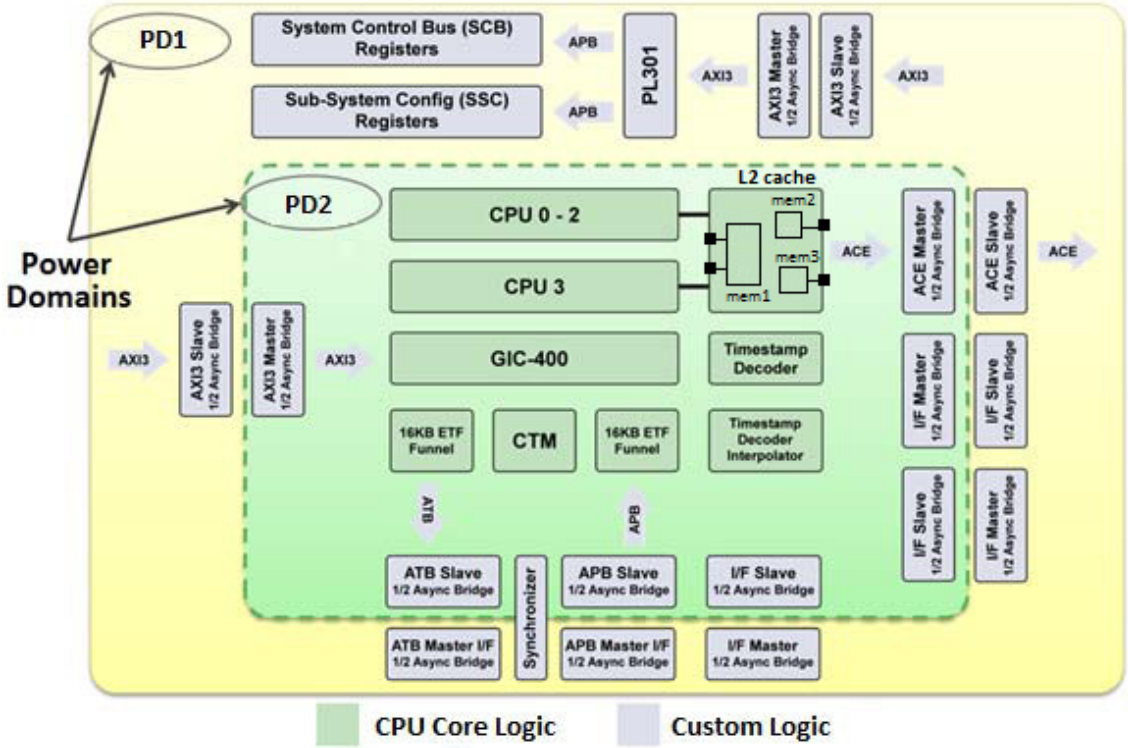
  Alternatively, use the -param *<name>=value>* argument of the uniquify_instance Tcl command to override the parameter value. Note that non integer type and dependent parameters are not allowed.

Swapping Memories

The CPU-core RTL delivered from third-party IPs contains instances of generic memories (defined through Verilog *always* blocks). These instances can be present inside the generate if and generate for blocks, which are often parameterized. See Example of RTL Containing Generic Memories.

When this CPU-core RTL is customized for end-application SoCs, these generic memories need to be replaced with more technology-specific memories to meet the objectives, such as power, performance, and area of end SoC applications.

The following figure shows the typical structure of a third-party IP containing various memories that are replaced with other memories to meet SoC requirements:



After memory swapping, additional pins may get generated, which should be propagated across hierarchies so that they can be controlled from a memory controller assembled at the IP sub-system level. For information on managing these pins, see Propagating Memory Pins Across Hierarchies After Memories Swap.

Example of RTL Containing Generic Memories

The following example shows the RTL containing generic memories within generate blocks (the code in green refer to memories):

```
generate //start of generate blockif((a_width < 8) || (b_width < 8))
begin: mult1MEM_1 #(a_width,b_width) u1(addr_bits, data_bits,
reset);endelse if ((a_width > 8) || (b_width > 8)) begin: mult2MEM_2
#(a_width,b_width) u2(addr_bits, data_bits, reset);MEM_1
#(a_width,b_width) u1(addr_bits, data_bits, reset);endelse begin: mult3 /
/3rd blockMEM_2 #(8,b_width) u3(addr_bits, data_bits, reset);MEM_1
#(a_width,b_width) u1(addr_bits, data_bits, reset);MEM_2
#(a_width,b_width) u2(addr_bits, data_bits, reset);MEM_2
#(a_width,b_width) u222(addr_bits, data_bits, reset);MEM_1
#(a_width,b_width) u22(addr_bits, data_bits, reset);endendgenerate
```

To see the complete example of performing memory swapping, see Example of Memory Swapping.

Types of Memory Swapping

Swapping memories is not a one-time activity because you often need to perform multiple swaps while trying to understand the correct configuration so that the end-SoC application meets the objectives, such as power, performance, and area.

You can perform the following types of memory swaps:

- Swapping One memory with One Technology Memory
- Swapping One Memory with Multiple Technology Memories
- Swapping Multiple Memories with One Technology Memory

On swapping memories, additional pins may get generated. For information on managing these pins, see Propagating Memory Pins Across Hierarchies After Memories Swap.

Swapping One memory with One Technology Memory

Perform this type of swapping by using the Update Masters functionality.

Swapping One Memory with Multiple Technology Memories

Perform the following steps for this type of swapping:

1. Group multiple memory instances into one memory group by using the group_design Tcl command.
2. Swap the single memory with the memory group by using the Update Masters functionality.

Swapping Multiple Memories with One Technology Memory

Perform the following steps for this type of swapping:

1. Group multiple memory instances into one memory group by using the group_design Tcl command.
2. Control the number of ports created on the memory group by using the min_ports argument of the infer_interfaces Tcl command.
3. Control the port names so that they are not dependent on glue logic drivers by using the *inmodule* port naming convention of the set_group_options Tcl command.
4. Change the port names of the group into meaningful names by using the update_master_interface Tcl command.

5. Swap the single memory with the memory group by using the Update Masters functionality.

Propagating Memory Pins Across Hierarchies After Memories Swap

While performing memory swapping, additional pins may get generated, which should be propagated across hierarchies so that they can be controlled from a memory controller assembled at the IP sub-system level.

To connect these additional memory pins to memory controllers which could be in another hierarchy, use the *Autoconnect* feature of GenSys. For details, see Using the Autoconnect Feature to Create Connections.

Using this feature, you can select all the unconnected memory pins and export them to the other design levels. The pin names should be derived from the various memory pin names by using the Autoconnect capturing method.

Note that you should select all the memory instance pins that are unconnected and export them with the minimum set of rules using capturing.

Swapping Memories Inside Generate Blocks

Before you swap memories inside a generate block, ensure the following:

- Set the -preserve_if_generate argument of the set_rtlimport_option Tcl command to true.
- Specify the modules in which memory swapping should be allowed if those modules contain the generate blocks.

  Specify these modules by using the set_ifgenerate_options Tcl command.
- Ensure that the generate blocks do not contain any glue logic or keywords other than if, else, begin, and end.

  Basically, the generate blocks should not have any assign or always or any other RTL that can be considered as glue by Gensys.

After performing the above actions, perform memory swapping as described in Types of Memory Swapping.

Generate Blocks in which Memory Swapping is Allowed

Memory swapping is allowed in the following types of generate blocks:

- generate if blocks that have if, else if, and else parts, as shown in the following example:

```
generate if(sel == 1'b1)A a();else if (sel_flavor_2 == 1'b1)B b();else
C c();endgenerate
```

- generate blocks that do not have any begin keyword to provide the region of generate block or an if block (generate if style). See the following example for the same:

```
generate if(sel == 1'b1)A a();endgenerate
```

- generate begin style blocks, as shown in the following example:

```
generate beginif(sel == 1'b1)A a();endendgenerate
```

- generate if begin style blocks, as shown in the following example:

```
generate if(sel == 1'b1) beginA a();endendgenerate
```

- generate begin if begin style blocks, as shown in the following example:

```
generate beginif(sel == 1'b1) beginA a();endendendgenerate
```

Example of Memory Swapping

This example shows swapping of two memories inside a generate block (the code in green refers to two memories that are being updated).

This example contains the following files:

- scu_ram_a.v
- ram_a.v
- cell_inter_clkgate.v
- generic_ram.v
- Include files, such as inc.v, inc1.v, inc2.v, inc3.h, and inc4.v
- Tcl File Containing Commands to Perform Memory Swapping

scu_ram_a.v

```
`include "inc.v"`include "inc1.v"`include "inc2.v"`include "inc3.h"module scu_ram_a
`L1D_RAM_PARAM_DECL (  input  wire clk,  input  wire DFTSE,  input  wire
[`L1DC_SIZE_W-1:0] l1_dc_size_i,  input  wire ram_a_clken_i,  input  wire
[`SCU_L1D_ASSOC-1:0] ram_a_cpu0_en_i,  input  wire ram_a_cpu0_wr_i,  input  wire
[`SCU_RAM_A_ADDR_W-1:0] ram_a_cpu0_addr_i,  input  wire [`SCU_RAM_A_DATA_W-1:0]
ram_a_cpu0_wdata_i,  output wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu0_way0_rdata_o,
output wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu0_way1_rdata_o,  output wire
[`SCU_RAM_A_DATA_W-1:0] ram_a_cpu0_way2_rdata_o,  output wire [`SCU_RAM_A_DATA_W-1:0]
ram_a_cpu0_way3_rdata_o,  input  wire [`SCU_L1D_ASSOC-1:0] ram_a_cpu1_en_i,  input
wire ram_a_cpu1_wr_i,  input  wire [`SCU_RAM_A_ADDR_W-1:0] ram_a_cpu1_addr_i,  input
wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu1_wdata_i,  output wire [`SCU_RAM_A_DATA_W-1:0]
ram_a_cpu1_way0_rdata_o,  output wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu1_way1_rdata_o,
output wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu1_way2_rdata_o,  output wire
[`SCU_RAM_A_DATA_W-1:0] ram_a_cpu1_way3_rdata_o,  input  wire [`SCU_L1D_ASSOC-1:0]
ram_a_cpu2_en_i,  input  wire ram_a_cpu2_wr_i,  input  wire [`SCU_RAM_A_ADDR_W-1:0]
ram_a_cpu2_addr_i,  input  wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu2_wdata_i,  output
wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu2_way0_rdata_o,  output wire
[`SCU_RAM_A_DATA_W-1:0] ram_a_cpu2_way1_rdata_o,  output wire [`SCU_RAM_A_DATA_W-1:0]
ram_a_cpu2_way2_rdata_o,  output wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu2_way3_rdata_o,
input  wire [`SCU_L1D_ASSOC-1:0] ram_a_cpu3_en_i,  input  wire ram_a_cpu3_wr_i,  input
wire [`SCU_RAM_A_ADDR_W-1:0] ram_a_cpu3_addr_i,  input  wire [`SCU_RAM_A_DATA_W-1:0]
ram_a_cpu3_wdata_i,  output wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu3_way0_rdata_o,
output wire [`SCU_RAM_A_DATA_W-1:0] ram_a_cpu3_way1_rdata_o,  output wire
[`SCU_RAM_A_DATA_W-1:0] ram_a_cpu3_way2_rdata_o,  output wire [`SCU_RAM_A_DATA_W-1:0]
ram_a_cpu3_way3_rdata_o);  wire [NUM_CPUS-1:0]              clk_tagram;  wire
[`SCU_RAM_A_ADDR_W-1:0]         cpu0_addr;  wire [`SCU_RAM_A_ADDR_W-1:0]
cpu1_addr;  wire [`SCU_RAM_A_ADDR_W-1:0]         cpu2_addr;  wire
[`SCU_RAM_A_ADDR_W-1:0]         cpu3_addr;  assign cpu0_addr = ram_a_cpu0_addr_i &
{l1_dc_size_i,{(`SCU_RAM_A_ADDR_W-`L1DC_SIZE_W){1'b1}}};  generate if (NUM_CPUS > 2)
begin : g_l1d_cpu0_rams  `GEN_RAM `TOP_SUB_PARA    u_ram_a_cpu0_way1  (     .clk
(clk_tagram[0]),      .addr_i          (cpu0_addr),      .rd_o
(ram_a_cpu0_way1_rdata_o),     .wd_i          (ram_a_cpu0_wdata_i),     .cs_i
(ram_a_cpu0_en_i[1]),      .we_i            (ram_a_cpu0_wr_i)    ); end else if
(NUM_CPUS == 2) begin   generic_ram `TOP_SUB_PARA    u_ram_a_cpu0_way2 (     .clk
(clk_tagram[0]),      .addr_i          (cpu0_addr),      .rd_o
(ram_a_cpu0_way2_rdata_o),     .wd_i          (ram_a_cpu0_wdata_i),     .cs_i
(ram_a_cpu0_en_i[1]),      .we_i            (ram_a_cpu0_wr_i)    );end else begin:
ending  `GEN_RAM `TOP_SUB_PARA `CPU0WAY3   (      .clk            (clk_tagram[0]),
.addr_i          (cpu0_addr),      .rd_o            (ram_a_cpu0_way3_rdata_o),
.wd_i            (ram_a_cpu0_wdata_i),      .cs_i            (ram_a_cpu0_en_i[1]),
.we_i            (ram_a_cpu0_wr_i)    );  end endgenerateendmodule
```

ram_a.v

```
module ram_a #(parameter                addr_bits         = 12,
data_bits         = 84,                 we_size           = 4,
reset             = 0,                  reset_high        = 0                  )(
input  wire                                clk,                                 input
wire [addr_bits-1:0]                addr_i,                   output wire
[data_bits-1:0]                  rd_o,                     input  wire
[data_bits-1:0]              wd_i,                 input  wire
cs_i,                input  wire [data_bits/we_size-1:0]        we_i,
input  wire                     ram_initn,           input  wire
ram_high_speed,           input  wire
ram_low_voltage,          input  wire [2:0]
ram_read_margin,          input  wire [1:0]
ram_write_margin,         input  wire
ram_input_gating,        input  wire                                ram_sleep,
input  wire                      ram_stdby,          input  wire
bypass,  input  wire [6:0]                        TA,  input  wire
TBIST,  input  wire                          TCSN,  input  wire
TED,  input  wire                            TOD,  input  wire
TWEN,  input  wire                           ATP,  input  wire
SE);wire               write_en;             wire [data_bits-3:0]   strb_i;
assign write_en =  |we_i ;genvar j;generate for (j = 0; j < data_bits-2; j = j +
1)begin: g_strb  assign strb_i[j]  = cs_i ? we_i[j/we_size] : 1'b0;end endgeneratewire
[data_bits-1:2] rd_o_int;assign rd_o[data_bits-1:2] = bypass ? wd_i[data_bits-1:2]
:rd_o_int[data_bits-1:2];assign rd_o[1:0] =  2'b0;SPL1CACHE_HIPERF_128x31m2_aTMol
u_ram_a (   .CK     (clk),   .A      (addr_i[6:0]),   .Q
(rd_o_int[data_bits-1:2]),   .D     (wd_i[data_bits-1:2]),   .CSN    (!cs_i),   .WEN
(!write_en),   .INITN  (ram_initn),   .HS     (ram_high_speed),   .LS
(ram_low_voltage),   .RM     (ram_read_margin),   .WM     (ram_write_margin),   .IG
(ram_input_gating),   .SLEEP  (ram_sleep),   .STDBY  (ram_stdby),   .TA     (TA   ),
.TBIST  (TBIST),   .TCSN   (TCSN ),   .TED    (TED  ),   .TOD    (TOD  ),   .TWEN
(TWEN ),   .ATP    (ATP),   .SE     (SE ),   .SCTRLI (1'b0),   .SCTRLO (),   .SDLI
(1'b0),   .SDLO   (),   .SDRI   (1'b0),   .SDRO   ()  );endmodule
```

cell_inter_clkgate.v

```
module cell_inter_clkgate (  input  wire clk_i,  input  wire clk_enable_i,  input
wire clk_senable_i,  output wire clk_gated_o);  wire clk_en;  reg  clk_en_reg;  assign
clk_en = clk_enable_i | clk_senable_i;  always @(clk_i or clk_en)     begin      if
(clk_i == 1'b0)        clk_en_reg <= clk_en;    end  assign clk_gated_o = clk_i &
clk_en_reg;endmodule
```

generic_ram.v

```
`include "inc.v"module generic_ram `para_over   (  input                          clk,
input  [addr_bits-1:0]         addr_i,  output [data_bits-1:0]         rd_o,  input
[data_bits-1:0]         wd_i,  input                          cs_i,  input
[data_bits/we_size-1:0] we_i);  reg [data_bits-1:0]   rdata_reg;  reg [data_bits-1:0]
mem[(1<<addr_bits)-1:0];  integer                i;  initial    if (reset)      for (i
= 0; i < (1<<addr_bits); i = i+1)        mem[i] = {data_bits{reset_high ? 1'b1 :
1'b0}};  wire   read_cycle  = cs_i & (~|we_i);  wire   write_cycle = cs_i & (|we_i);
wire [data_bits-1:0] real_we;  genvar j;  generate for (j = 0; j < data_bits; j = j +
1) begin : g_we    assign real_we[j] = cs_i ? we_i[j/we_size] : 1'b0;  end endgenerate
wire [data_bits-1:0] nxt_data = (~real_we & mem[addr_i]) | ( real_we & wd_i);  always
@ (posedge clk)    if (write_cycle)      mem[addr_i] <= nxt_data;  always @ (posedge
clk)    if (read_cycle)      rdata_reg <= mem[addr_i];    else      rdata_reg <=
{data_bits{1'bx}};  assign rd_o = rdata_reg;endmodule
```

mem_a.f

```
scu
```

```
generic_ram.v
cell_inter_clkgate.v
-enable_precompile_vlog-mthresh 600-stop cell_inter_clkgate-param
scu_l1d_tagrams.CPU_CACHE_PROTECTION=0-param
scu_l1d_tagrams.L1_DCACHE_SIZE="3'b011"-param
scu_l1d_tagrams.NUM_CPUS=4-param scu_l1d_tagrams.SCU_CACHE_PROTECTION=0
```

inc.v

```
`define para_over  #(parameter addr_bits  = 2,  \
parameter data_bits  = 4,  \                    parameter we_size    = 1,
\                   parameter reset      = 0,  \
parameter reset_high = 0) `define L2_SIZE_W     2`define SCU_L1D_ASSOC
2`define SCU_RAM_A_ADDR_W 4`define SCU_RAM_A_DATA_W (CPU_CACHE_PROTECTION
? 40 : 33)`define L1DC_SIZE_W    3 `define L1D_RAM_PARAM_DECL #(parameter
NUM_CPUS   = 4,  \                  parameter L1_DCACHE_SIZE       =
3'b011, \            parameter SCU_CACHE_PROTECTION = 1'b0,  \
parameter CPU_CACHE_PROTECTION = 1'b0)`define L1D_RAM_PARAM_INST
#(.NUM_CPUS          (NUM_CPUS),           \
.L1_DCACHE_SIZE      (L1_DCACHE_SIZE),       \.SCU_CACHE_PROTECTION
(SCU_CACHE_PROTECTION), \.CPU_CACHE_PROTECTION
(CPU_CACHE_PROTECTION))`include "inc4.v"
```

inc1.v

```
`define  TOP_SUB_PARA \ #(\      .addr_bits  (`SCU_RAM_A_ADDR_W), \
.data_bits  (`SCU_RAM_A_DATA_W), \     .we_size    (`SCU_RAM_A_DATA_W) \
)
```

inc2.v

```
`define CPU0WAY0   u_ram_a_cpu0_way0`define CPU0WAY1
u_ram_a_cpu0_way1`define CPU0WAY3     u_ram_a_cpu0_way3
```

inc3.h

```
`define CPU1WAY0     u_ram_a_cpu1_way0`define CPU2WAY0
u_ram_a_cpu2_way0
```

inc4.v

```
`define GEN_RAM generic_ram
```

Tcl File Containing Commands to Perform Memory Swapping

```
set_rtlimport_option -preserve_if_generate true
set_ifgenerate_options -preserve_component { generic_ram }
load_rtl -enableSV -f mem_a.f -top scu_ram_a +incdir+inc+inc/inc1+inc/
inc2+inc/inc3+inc/inc4+
load_rtl -component ram_a.v
select_design scu_ram_a
update_master -instance u_ram_a_cpu0_way1 -name ram_a
update_master -instance u_ram_a_cpu0_way2 -name ram_a
run GenerateVerilog
```

Editing generate Blocks and Array-Of-Instances

While modifying an RTL by using GenSys, the designer may need to replace third-party memories with an internal memory. The RTL for memories is usually present within the generate if, generate-for, or array-of-instances RTL statements. Each memory has a master associated.

To replace a third-party memory instance in these RTL statements with an internal memory, you need to update the master of that memory instance with a new master, which is the master of an internal memory. For details, see Updating the Master of Instances.

Once you update master, some additional pins get generated on the *generate* or array boundaries. You can tieoff such hanging pins or export them to the design boundary. For details, see Adding Tieoff Connections to Instances and Exporting Instance Pins.

You can also delete any unwanted connections from memory pins. For details, see Exporting Complex Multi-Instance Pins.

Prerequisites to Edit Generate Blocks and Array-Of-Instances

To allow editing of these structures, set the -preserve_generate argument of the set_rtlimport_options Tcl command to TRUE.

Multi-Instance

Within generate-for and array-of-instances structures, various instances are controlled by a loop variable. GenSys clubs all such instances in one group known as multi-instance.

When you make any change to a multi-instance, GenSys automatically applies that change to all the instances within that multi-instance. GenSys does not allow you to edit any individual instance within a multi-instance.

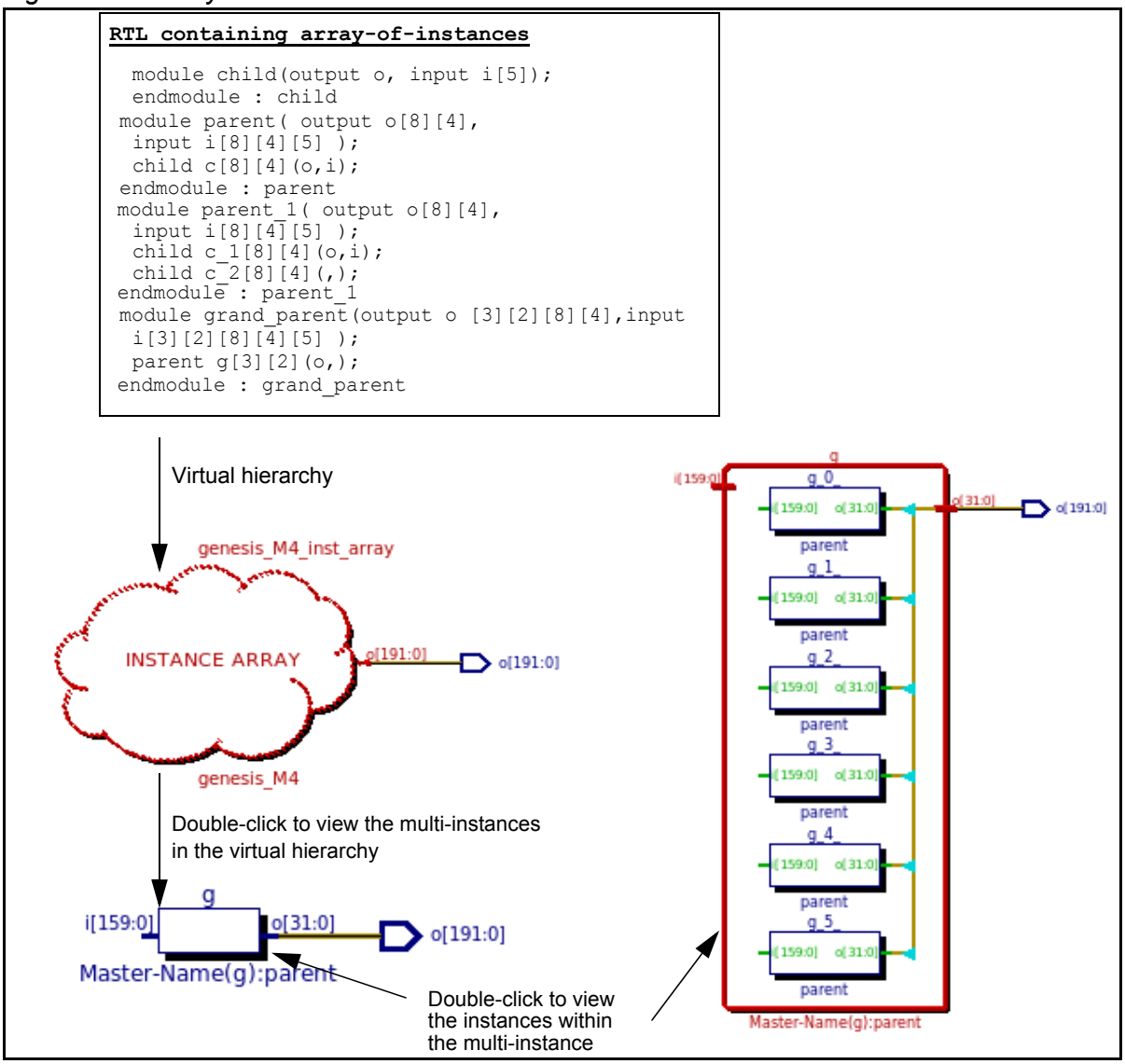
array-of-instances

An array-of-instances appears in a cloud-like virtual hierarchy. Double-click on the cloud to view the Multi-Instance, which is the consolidated view of all the instances inside the array. Double click the Multi-Instance to view all the instances in a multi-instance.

Any update made to a Multi-Instance automatically updates all the instances within that hierarchy. You cannot edit individual instances under a Multi-Instance.

See the following figure showing a multi-instance and its instances:

Figure 6-1 Array-of-instances

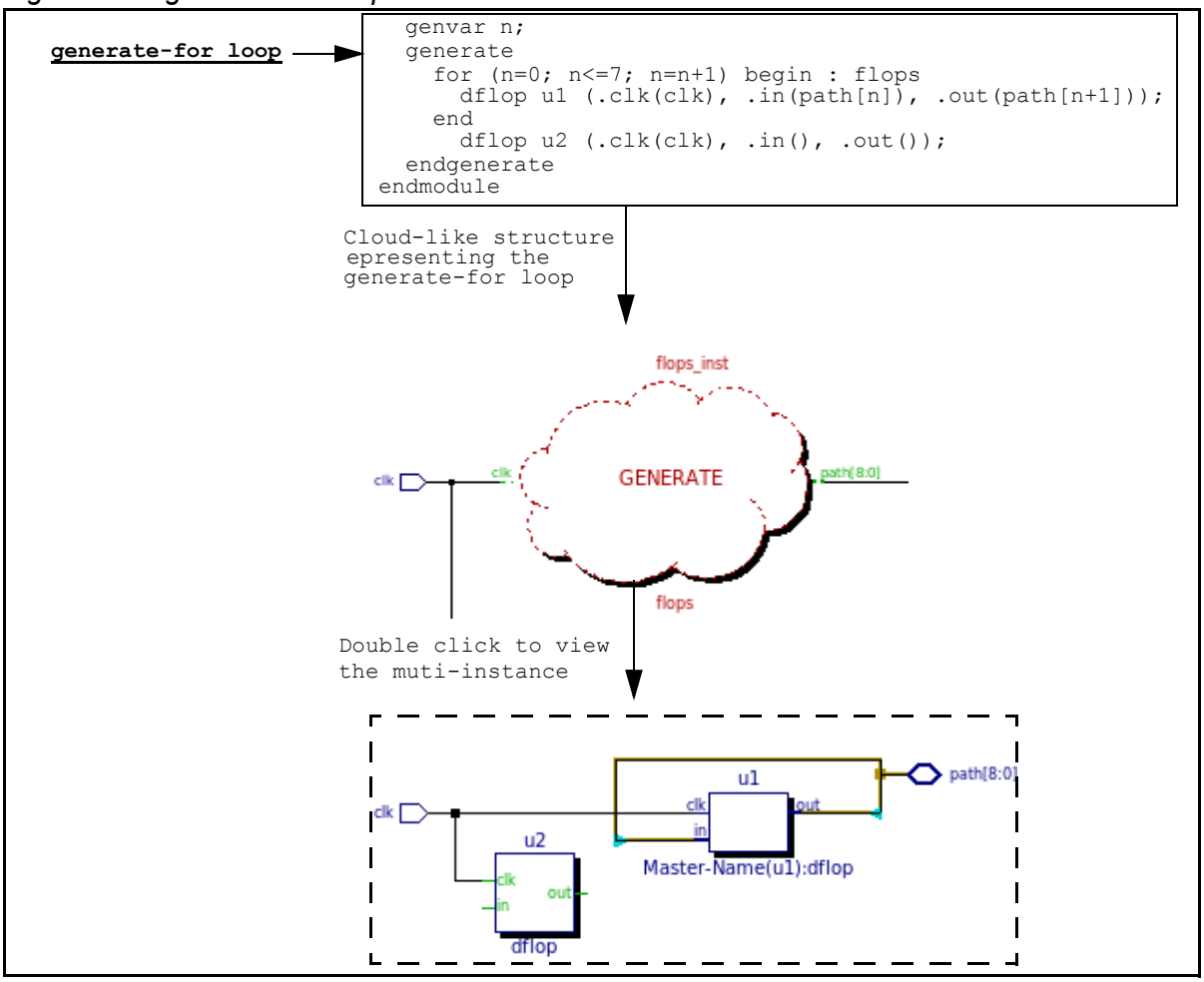


generate-for

Instances generated by a *generate-for* loop appear in a cloud-like virtual hierarchy. Double-click on this hierarchy to view the Multi-Instance, which is the consolidated view of all the instances generated by the *generate-for* loop.

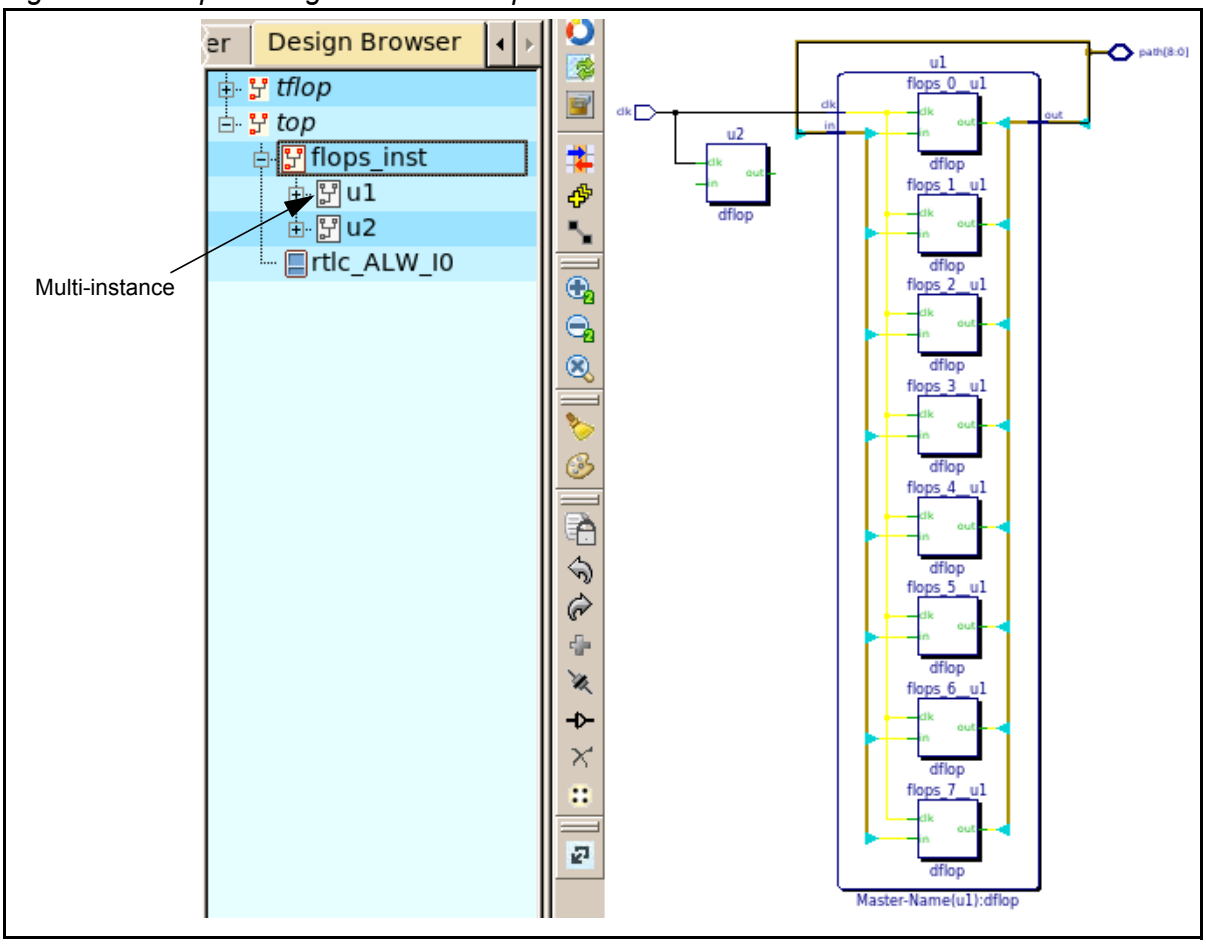
Any update made to a Multi-Instance automatically updates all the instances within that hierarchy. You cannot edit individual instances under a Multi-Instance. See the following figure showing a multi-instance:

Figure 6-2 generate-for loop



When you double-click on a Multi-Instance, GenSys shows all the instances within that Multi-Instance. This is shown in the following figure:

Figure 6-3 Expanded generate-for loop



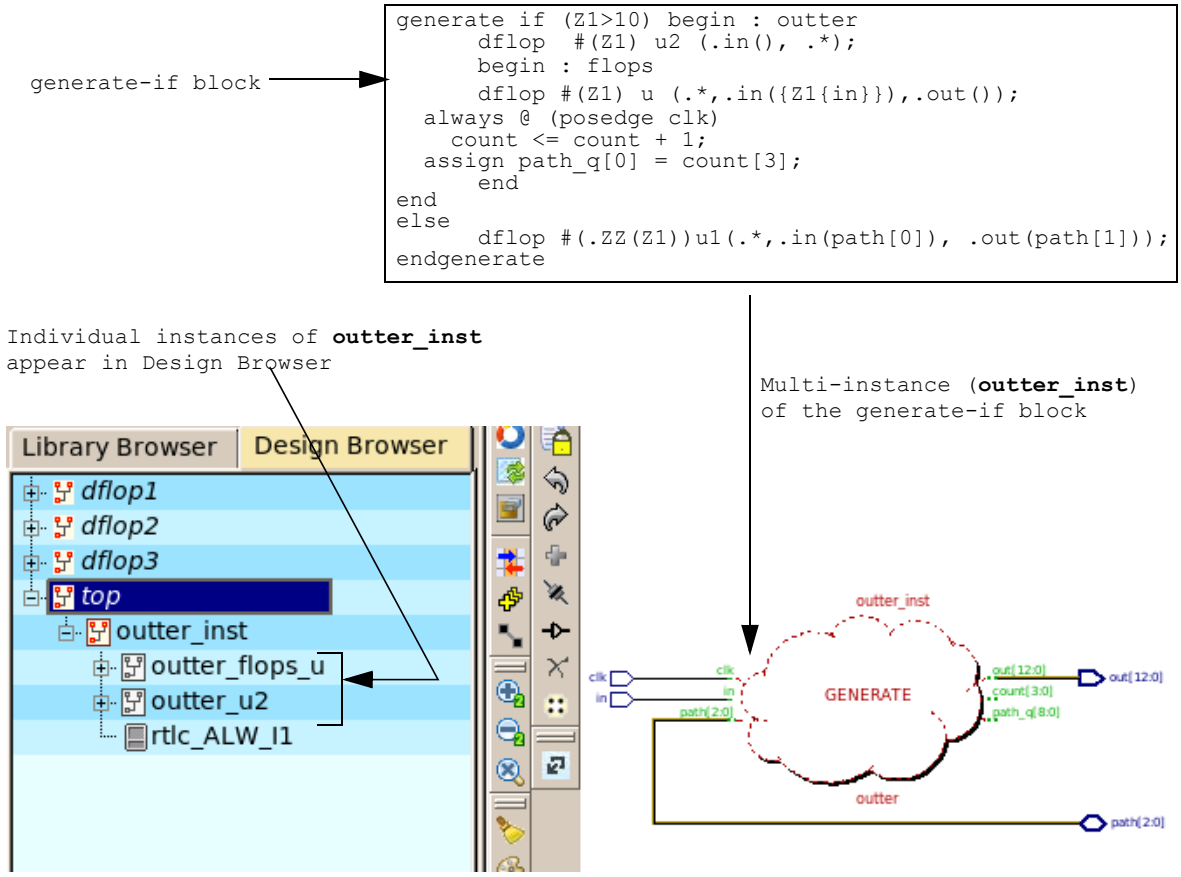
generate if

Instances generated by a *generate-if* loop appear in a cloud-like virtual hierarchy. When you double-click on this hierarchy, all the instances of that hierarchy appear in the schematic.

In addition, the *Design Browser* shows all the instances of a *generate-if* loop. The condition under which an instance is visible is shown in *the Generate-If Condition* row of the *Property Editor* pane.

The following figure shows the instances of a *generate-if* block:

Figure 6-4 generate-if block example



In the above example, if you double-click on the virtual hierarchy (outter_inst), all the instances within outter_inst appear that you can edit individually.

Updating the Master of Instances

The update master operation replaces the instance master with a new master. This operation is useful for memory swapping when you want to replace a third-party memory with a different memory.

For an array-of-instances and a generate-for loop, this operation updates the master of all the instances inside a Multi-Instance. You cannot perform this operation on individual instances.

However, for the generate if loop, you can update the master of individual instances.

You can update master in any of the following ways:

- Tcl

  Use the update_master Tcl command to specify the details of the new master.

- GUI

  Right-click on the instance and select the Update Master option from the shortcut menu. The Update Masters dialog appears in which you can specify the VNLV information of the new master.

  For information on this dialog, refer to the *GenSys User Guide*.

See Example of Updating the Master of a generate-for Multi-Instance.

Example of Updating the Master of a generate-for Multi-Instance

Consider the following RTL specified as input to GenSys:

```
module dflop(input clk,                input in,                output
reg out);always@(posedge clk)        out<=in;endmodulemodule tflop(input
clk,                input in,                output reg
out);always@(posedge clk)       out<=~in;endmodulemodule top (clk, out);
input        clk;  reg     [3:0] count ;  wire    [8:0] path;  output
out;  always @ (posedge clk)    count <= count + 1;  assign path[0] =
count[3];  assign out = path[8];  genvar n;  generate    for (n=0; n<=7;
n=n+1) begin : flops      dflop u1 (.clk(clk), .in(path[n]),
.out(path[n+1]));    end      dflop u2 (.clk(clk), .in(), .out());
endgenerateendmodule
```

In the above RTL, the *generate-for* loop creates the u1 Multi-Instance that contains eight instances. To update the master of u1, perform the following steps:

1. Specify the following Tcl command:

   ```
   set_rtlimport_option -preserve_generate TRUE
   ```

2. Load the RTL in GenSys by specifying the following Tcl command:

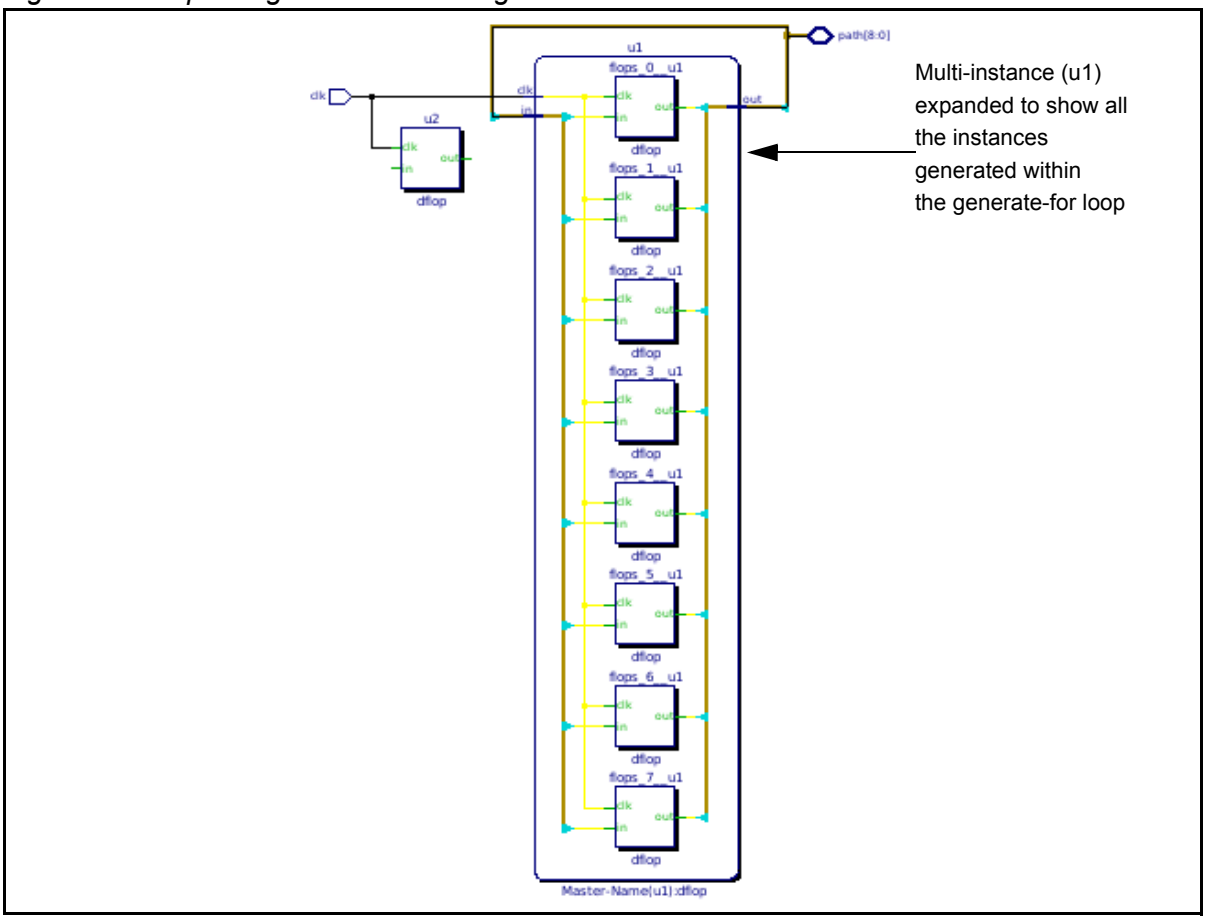
   ```
   load_rtl input.v
   ```

3. In the *Design Browser*, double-click on the top design to select this design. Alternatively, specify the following Tcl command:

   ```
   select_design top
   ```

   The following figure shows the expanded multi-instance (u1) in the top design:

Figure 6-5 Updating the master of a generate-for multi-instance



4. To update the master of the u1 Multi-Instance, specify the following Tcl command:

   ```
   update_master -instance u1 -name tflop
   ```

   This step updates the master from u1 to tflop for all the instances within u1.

5. Generate Verilog by specifying the following Tcl command:

   ```
   run GenerateVerilog
   ```

After performing the above steps, the generated Verilog contains the update master changes.

The generated Verilog is as follows:

```
module dflop(input clk,
             input in,
             output reg out);
always@(posedge clk)
      out<=in;
endmodule
```

```
module tflop(input clk,
             input in,
             output reg out);
always@(posedge clk)
      out<=~in;
endmodule
module top (clk,
          out);
input          clk;
output         out;
//
reg [3:0] count;
wire [8:0] path;
//
//
genvar n;
//
//
  generate
    for (n=0; n<=7; n=n+1) begin : flops
      tflop u1 (.clk(clk), .in(path[n]), .out(path[n+1]));
    end
      dflop u2 (.clk(clk), .in(), .out());
  endgenerate
//
//
  always @ (posedge clk)
   count <= count + 1;
  assign path[0] = count[3];
//
endmodule
```

Adding Tieoff Connections to Instances

GenSys allows adding connections of the tieoff type for array-of-instances and instances in generate-for and generate if.

When you add a tieoff connection on a Multi-Instance pin, GenSys automatically creates that connection from all the instances present inside that Multi-Instance.

For an array-of-instances and a generate-for loop, you cannot add tieoff connections to individual instances. However, for a generate if loop, you can add tieoff connections on selected instances.

You can add tieoff connections in any of the following ways:

- Tcl

  Use the add_connection Tcl command specified with the -tieoff argument. If you specify any other connection type other than tieoff, GenSys reports an error.

  For details on this command, refer to *GenSys Tcl Commands Reference Guide*.

- GUI

  To add a tieoff connection in GUI, perform the following steps:

  1. Right-click on the instance pin, and select the Add Tie-off option from the shortcut menu.

  The Add Tie-off dialog appears, as shown in the following figure:

Figure 6-6 The Add Tie-off dialog

  1. In the Add Tie-off dialog, specify a tieoff value.
  2. Click Add.

See Example of Adding a Tieoff Connection from a Multi-Instance.

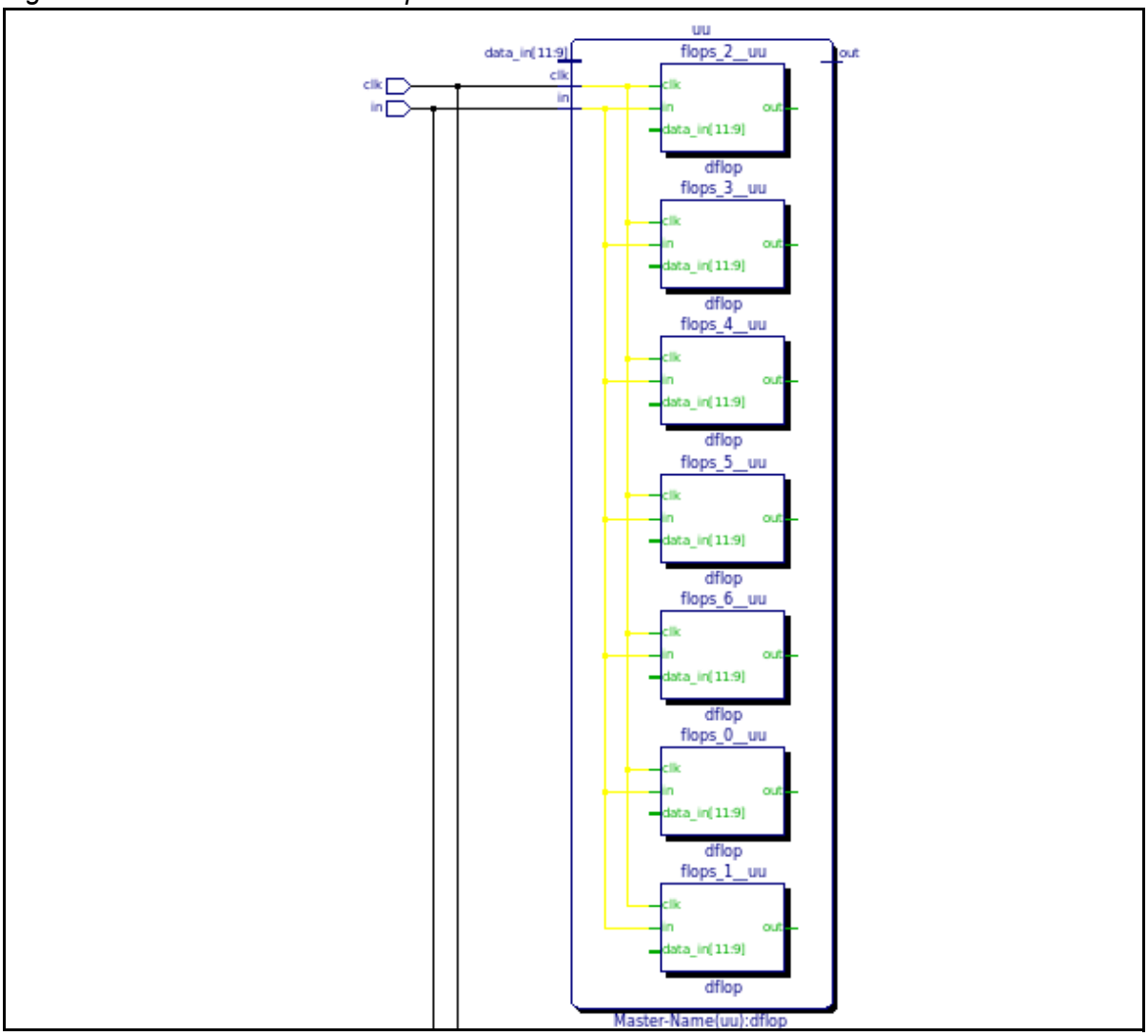
Example of Adding a Tieoff Connection from a Multi-Instance

Consider the following RTL specified as input to GenSys:

```
module top (clk, out,in);  input         clk,in;  reg     [3:0] count =
0;  wire    [8:0] path;  output        out;wire [7*2-1:0] data_in;
always @ (posedge clk)    count <= count + 1;  genvar n;  generate    for
(n=0; n<7; n=n+1) begin : flops      dflop u (.clk(clk), .in(in),
.out(path[n+1]),.data_in(data_in[2*n+1:2*n]));      dflop uu (.clk(clk),
.in(in), .out(),.data_in());    end  endgenerateendmodulemodule
dflop(input clk,  input in,  output reg out,input [11:9]data_in);
always@(posedge clk)  out<=in;endmodule
```

In the above RTL, the *generate-for* loop generates two Multi-Instances, u and uu. The following figure shows the uu Multi-Instance containing multiple instances generated by the *generate-for* loop:

Figure 6-7 Multi-instance example



In the above uu Multi-Instance, to create a tieoff connection from the data_in pin, perform the following steps:

1. Specify the following Tcl command:

```
set_rtlimport_option -preserve_generate TRUE
```

2. Load the RTL in GenSys by specifying the following Tcl command:

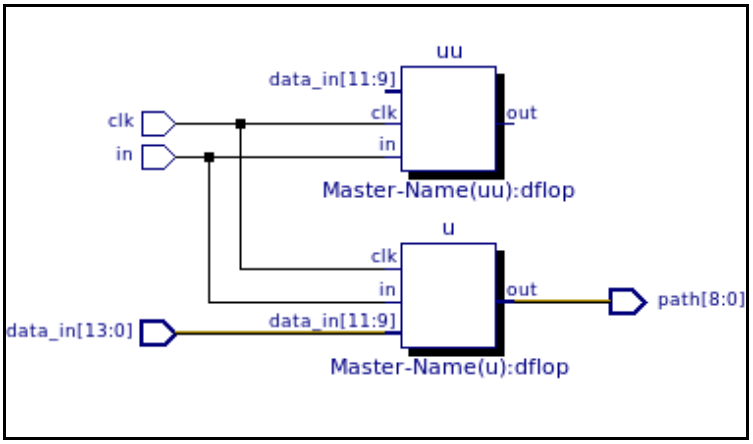
```
load_rtl input.v
```

3. Specify the following Tcl commands to select the flops design containing the uu Multi-Instance:

```
select_design top
select_design flops
```

The following schematic show the selected flops design in which the u and uu Multi-Instance is present:

Figure 6-8 Schematic of a multi-instance



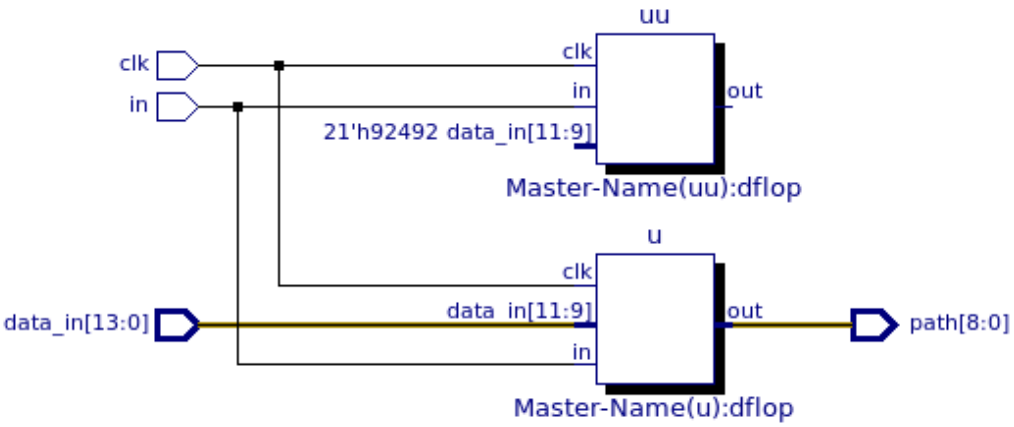
The expanded view of uu is shown in Figure 6-7.

4. Create a tieoff connection from the uu Multi-Instance by specifying the following Tcl command:

```
add_connection -instance uu -pin {data_in} -tieoff 2
```

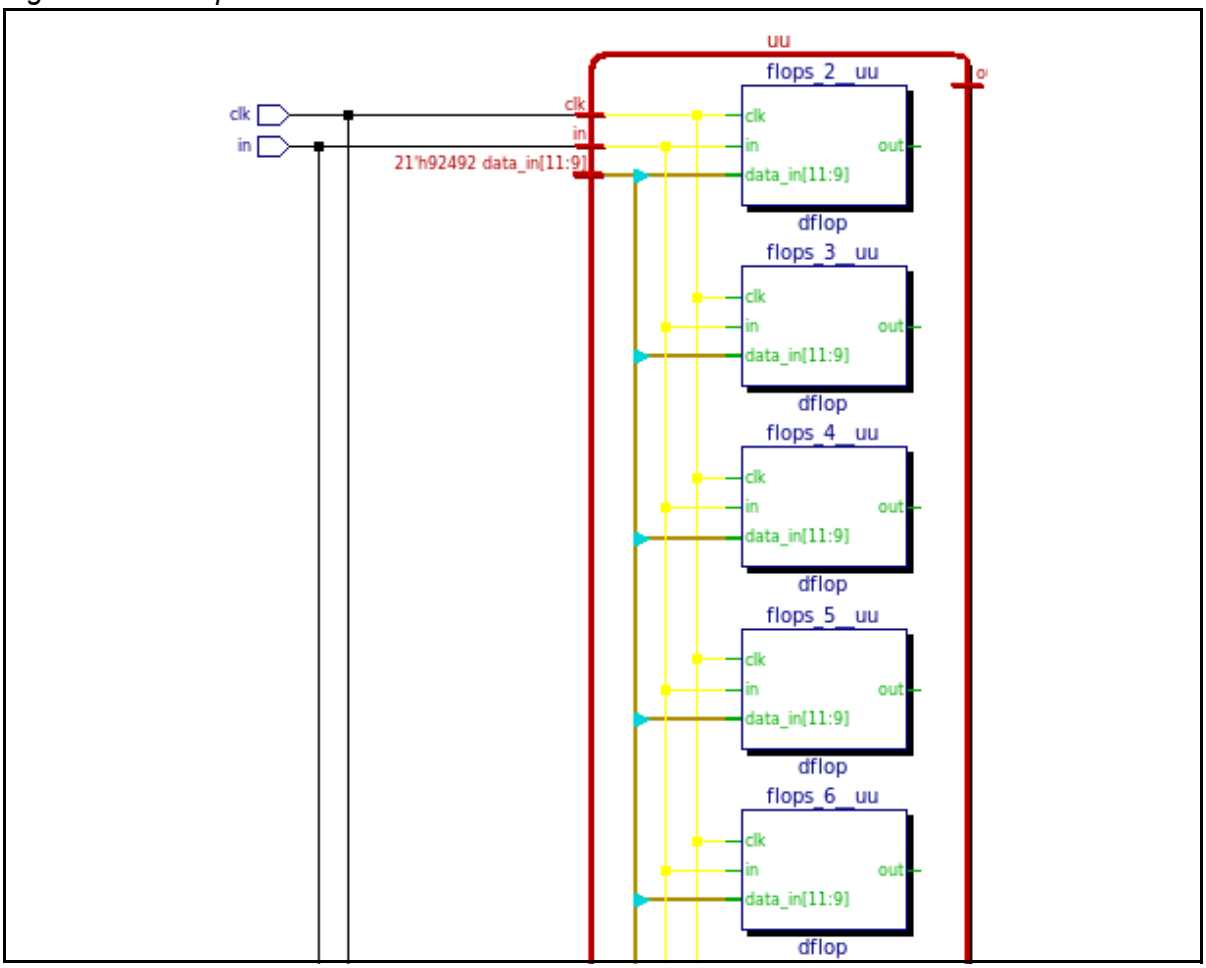
After performing this step, a tieoff connection appears in the data_in pin, as shown in the following figure:

Figure 6-9 Tie-off added to the multi-instance



If you expand the uu group by double-clicking it in the schematic, you will notice that a tieoff connection is created from the data_in pins of all the instances present in the uu Multi-Instance. This is shown in the following figure:

Figure 6-10 Expanded multi-instance with tie-off connections



Deleting Connections from Instances

When you delete a connection from a Multi-Instance pin, GenSys automatically deletes that connection from all the instances present inside that Multi-Instance.

For an array-of-instances and a generate-for loop, you cannot delete connections from individual instances. However, for a generate-if loop, you can delete connections from selected instances.

You can delete connections in any of the following ways:

- Tcl

  Use the delete_connection Tcl command. For information on this command, refer to the *GenSys Tcl Commands Reference Guide*.

- GUI

  Right-click on an instance pin and select the Delete Connected Net option from the shortcut menu.

Exporting Instance Pins

To export a pin from a Multi-Instance to the design boundary, use the following Tcl command:

```
export_pin -instance <multi-instance-name> -pin <pin-name>-export_name
<exported-pin-name> -loop_expr <expression>
```

The arguments of the above command are explained as follows:

- *-loop_expr <expression>*: This is the new argument.

  Use this argument to export complex Multi-Instance pins. For details, see Exporting Complex Multi-Instance Pins.

- *-instance <multi-instance-name>*: Specifies the name of the Multi-Instance.
- *-pin <pin-name>*: Specifies the Multi-Instance pin to be exported.
- *-export_name <exported-pin-name>*: Specifies the name of the pin by which it should be exported.

Any argument other than the above-mentioned arguments are invalid for a Multi-Instance. If you specify any invalid argument, GenSys reports an error.

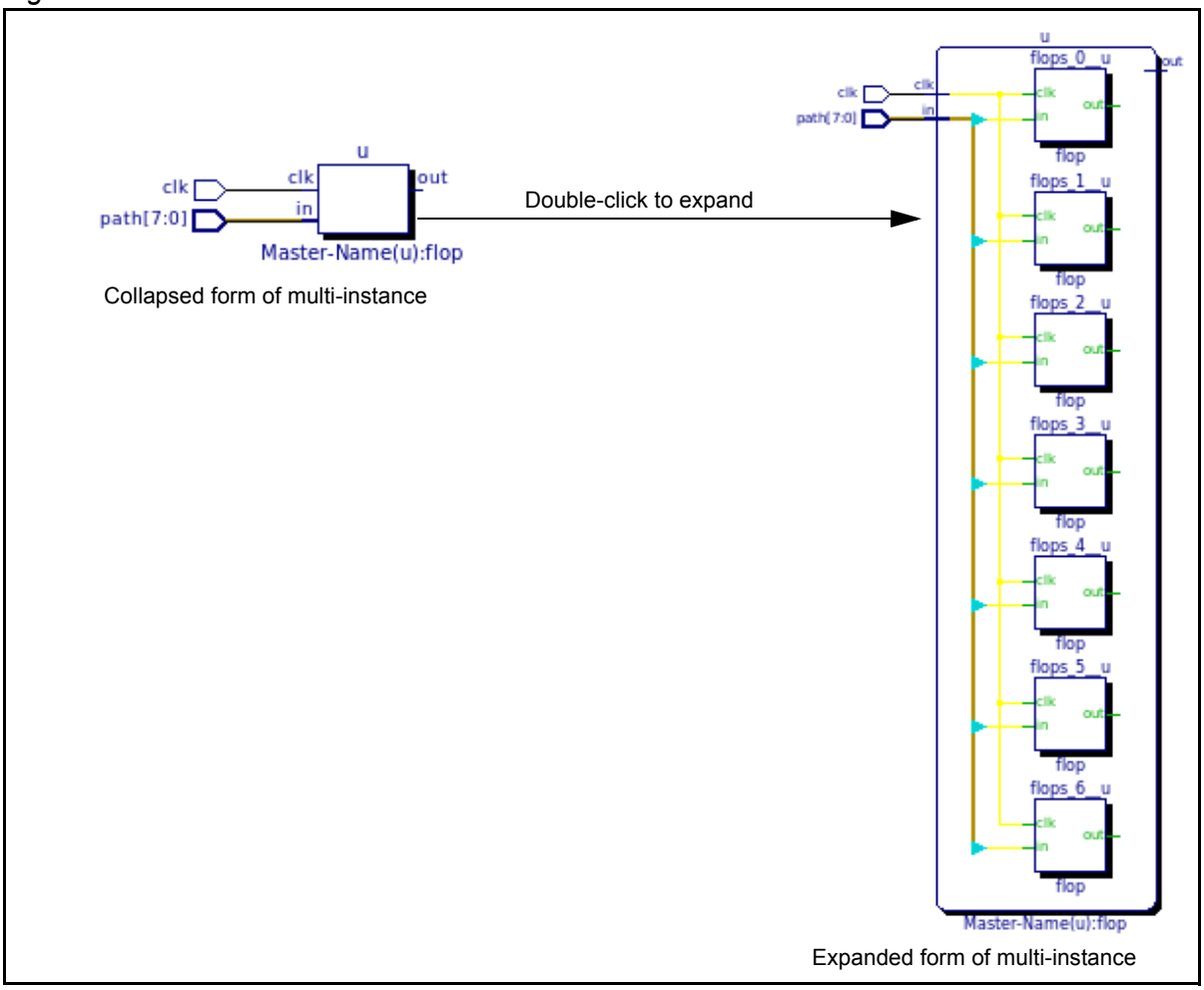
Example of Exporting a Multi-Instance Pin

Consider the following RTL specified as input to GenSys:

```
module top #(parameter Z0 = 0,parameter Z1 = 15,parameter Z2 = 5
)(clk,reset, out,in);  input         clk,reset;input [Z1:0] in;  output
logic [Z1*7-1:0] out [2] ;  genvar l,n,m;  wire [7:0] path;  generate
for (l=0; l<7; l=l+1) begin : flops      flop u (.clk(clk), .in(path[l]),
.out());    end  endgenerate  generate    for (n=0; n<7; n=n+1) begin :
dflops      dflop #(.Z0(Z0),.Z1(Z1))u1 (clk, in,
{out[0][Z1*n+7:Z1*n]});    end    for (m=0; m<7; m=m+1) begin : tflops
tflop #(.Z0(Z0),.Z1(Z1))u2 (clk,in,      out[1][Z1*m+7:Z1*m]);    end
endgenerateendmodulemodule dflop#(parameter Z0 = 0,parameter Z1 =
12)(input clk,input [Z1:Z0] in,output reg [Z1:Z0] out);always@(posedge
clk)out<=in;endmodulemodule tflop#(parameter Z0 = 0,parameter Z1 =
12)(input clk,input [Z1:Z0] in,output reg [Z1:Z0] out);always@(posedge
clk)out<=~in;endmodulemodule flop #(parameter Z1 = 12)(input clk,input
in,output reg out);always@(posedge clk)out<=in;endmodule
```

For the above example, the following figure shows the schematic of the u Multi-Instance and its expanded form when you double-click on u:

Figure 6-11 Multi-instance



In the above example, to export the out pin of the u Multi-Instance so that the ex_output pin appears on the design boundary, perform the following steps:

1. Specify the following Tcl command:

   ```
   set_rtlimport_option -preserve_generate TRUE
   ```

2. Load the RTL in GenSys by specifying the following Tcl command:

   ```
   load_rtl input.v -enableSV
   ```

3. Specify the following Tcl commands to select the flops design containing the u Multi-Instance:

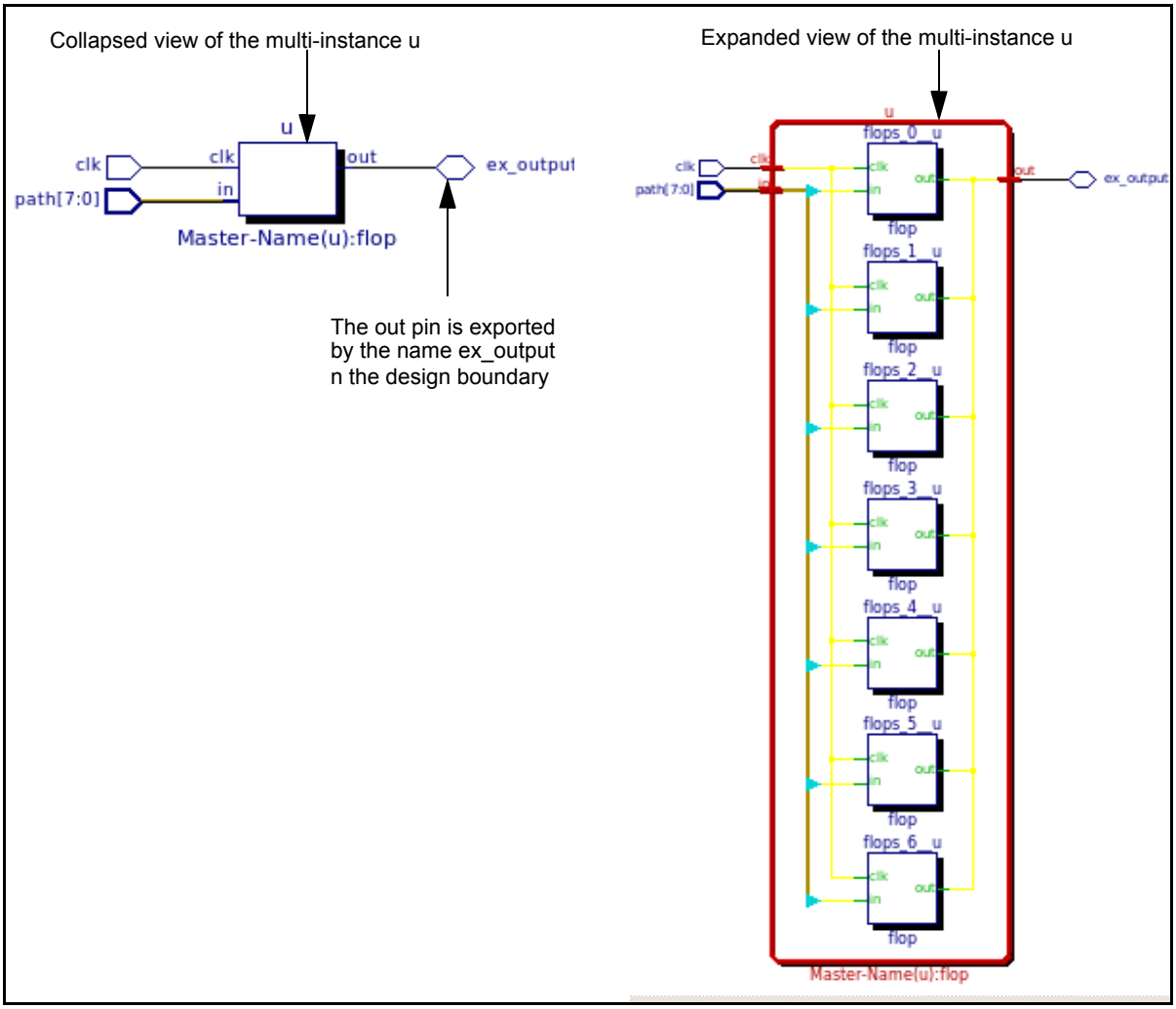
   ```
   select_design topselect_design flops
   ```

4. Export the out pin by specifying the following Tcl command:

   ```
   export_pin -instance u -pin out -export_name ex_output
   ```

After performing this step, the out pin of the u Multi-Instance is exported by the name ex_output on the design boundary. This is shown in the following figure:

Figure 6-12 Example of exporting pins of a multi-instance



5. Generate Verilog by using the following Tcl command:

```
run GenerateVerilog
```

After performing the above steps, GenSys generates a Verilog file in the *<curr-work-dir>/output/Verilog* directory. This file preserves the changes made to the u Multi-Instance.

The following Verilog file is generated for this example:

```
module top #(parameter    Z0   =    0,parameter    Z1   =    15,parameter
Z2   =    5)           (clk  ,            reset,            out  ,
in);input              clk;input              reset;output logic
[((Z1*7)-1):0] out [0:(2-1)]  ;input  wire [Z1:0]          in
;wire    [7:0]     path;wire              ex_output;//genvar l;genvar
n;genvar m;////  generate    for (l=0; l<7; l=l+1) begin : flops
flop u (.clk(clk), .in(path[l]), .out( ex_output));    end  endgenerate
////  generate    for (n=0; n<7; n=n+1) begin : dflops      dflop
#(.Z0(Z0),.Z1(Z1))u1 (clk, in,      {out[0][Z1*n+7:Z1*n]});    end    for
(m=0; m<7; m=m+1) begin : tflops    tflop #(.Z0(Z0),.Z1(Z1))u2 (clk,in,
out[1][Z1*m+7:Z1*m]);    end  endgenerate//endmodulemodule
dflop#(parameter Z0 = 0,parameter Z1 = 12)(input clk,input [Z1:Z0]
in,output reg [Z1:Z0] out);always@(posedge clk)out<=in;endmodulemodule
tflop#(parameter Z0 = 0,parameter Z1 = 12)(input clk,input [Z1:Z0]
in,output reg [Z1:Z0] out);always@(posedge clk)out<=~in;endmodulemodule
flop #(parameter Z1 = 12)(input clk,input in,output reg
out);always@(posedge clk)out<=in;endmodule
```

Exporting Complex Multi-Instance Pins

To connect complex multi-instance pins inside generate-for or array-of-instance, you must first export such pins on the generate-block boundary by using the export_pin Tcl command.

For such pins, you must specify an RTL expression by using the -loop_expr argument of the export_pin Tcl command. This is the new argument added in the 5.5 release. The RTL expression comprises of:

- The signal name.
- The genvar variables controlling the generate-for loop.

See Example of Specifying RTL Expression in -loop_expr.

If you do not use the -loop_expr argument, the corresponding pin of all the instances present inside Multi-Instance get connected to the same exported pin. This might be the desired behavior for clock pins but not for other pin types.

After exporting pins, you can create connections with the exported pins by using the add_connection Tcl command. For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

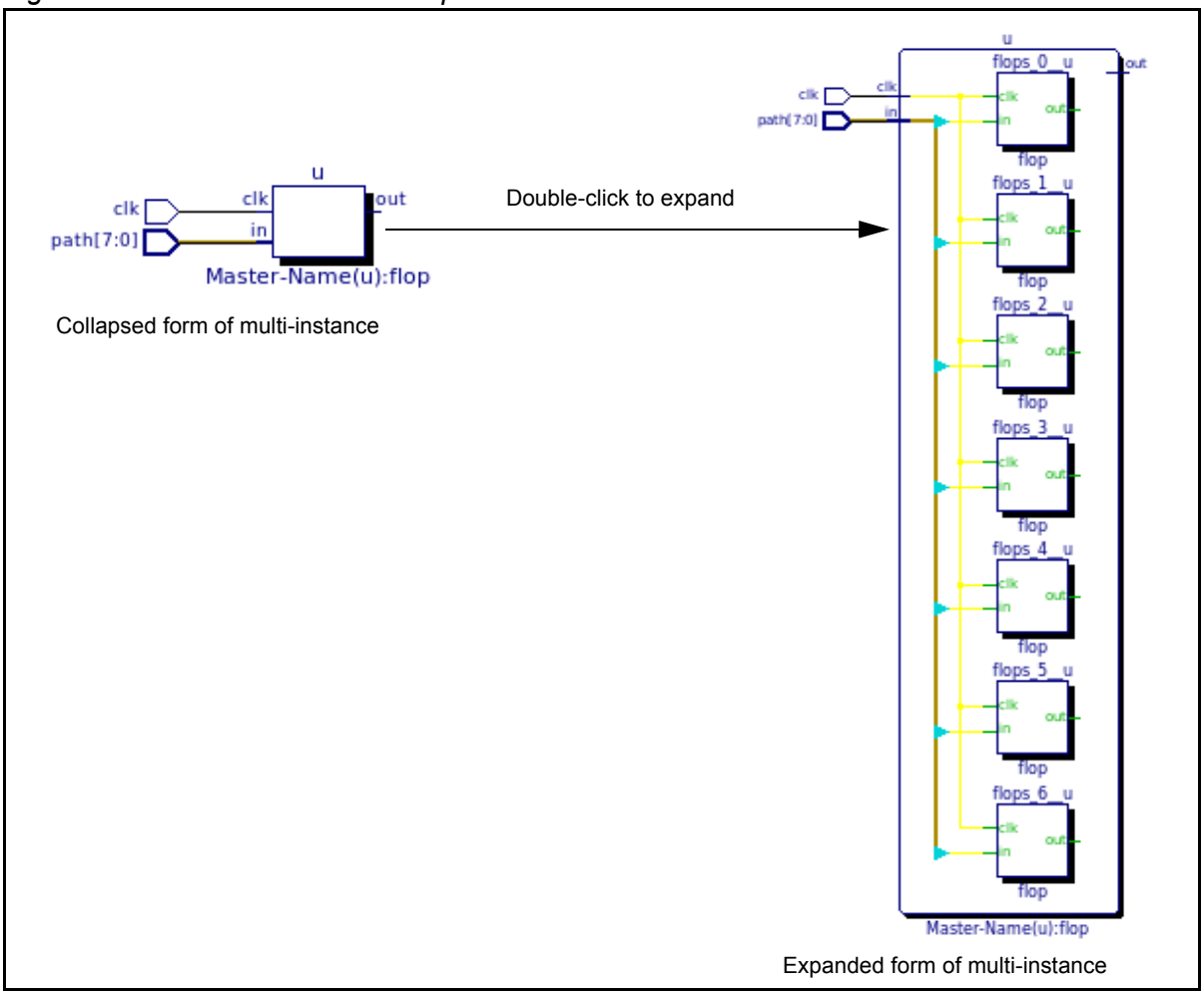
Example of Specifying RTL Expression in -loop_expr

Consider the following RTL snippet:

```
genvar l,n,m;  wire [7:0] path;  generate    for (l=0; l<7; l=l+1) begin
: flops      flop u (.clk(clk), .in(path[l]), .out());    end
endgenerate
```

The *for* loop in the above RTL generates eight instances that form the u Multi-Instance group, as shown below:

Figure 6-13 Multi-instance example



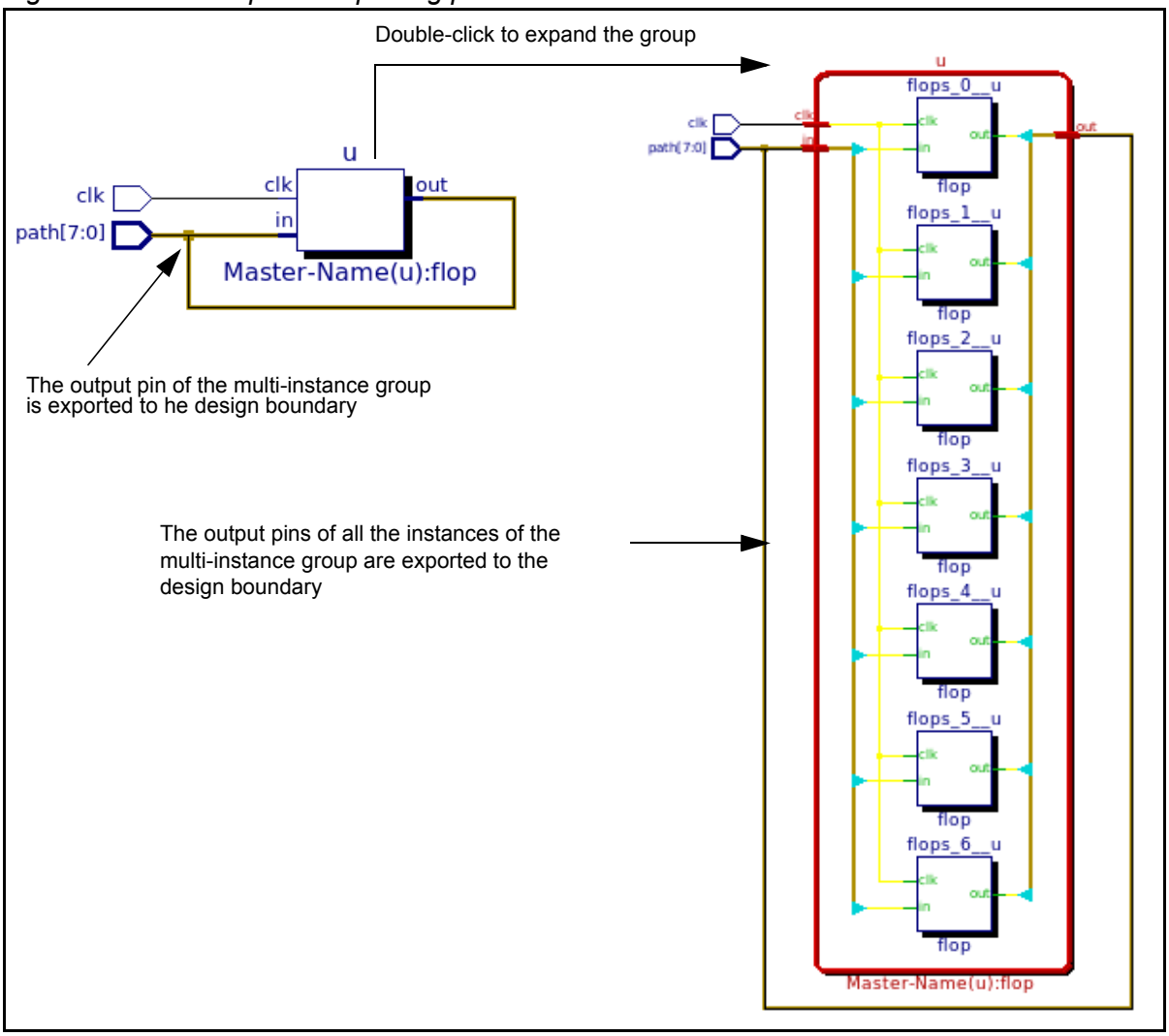
In the above example, to export the out pin of the u Multi-Instance, specify the expression {path[l+1]} to the -loop_expr argument, as shown in the following Tcl command:

```
export_pin -instance u -pin out -loop_expr {path[l+1]}
```

Note that expression in the above command contains the RTL variables, path and l.

The out pin gets exported to the design boundary, as shown in the following figure:

Figure 6-14 Example of exporting pins of a multi-instance



7

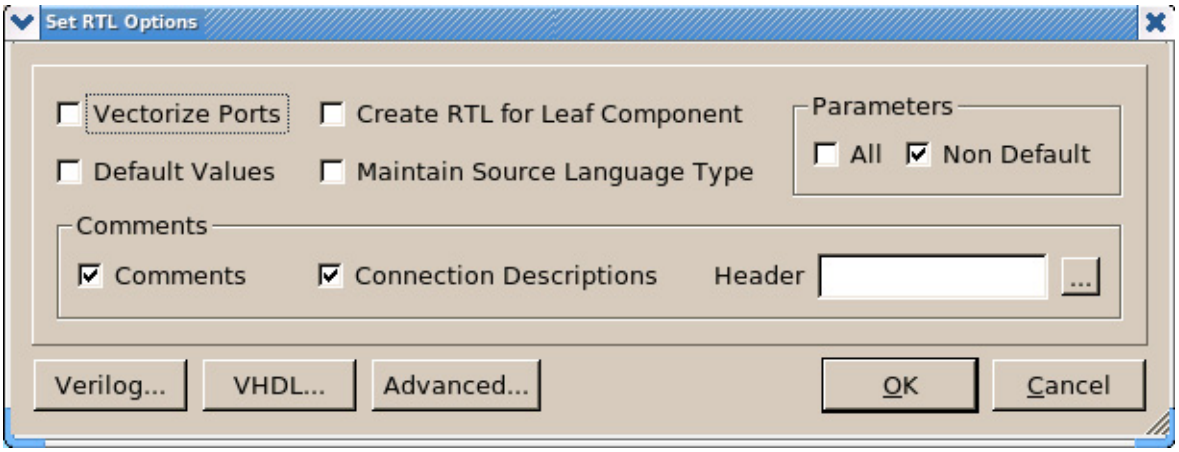
Generating RTL

Once you finalize the design, you can generate the RTL, such as Verilog or VHDL. You can generate RTL in any of the following ways:

- Click the *Generator -> GenerateVerilog* menu option to generate the Verilog files of the design.
- Click the *Generator -> GenerateVHDL* menu option to generate the VHDL files of the design.

On performing any of the above actions, the *Set RTL Options* dialog appears, as shown in the following figure:

Figure 7-1 The Set RTL Options Dialog



In the above dialog, select the options based on your requirements and then click the *OK* button to generate the RTL.

Adding Comments in the RTL

Set the -rtl_comment argument of the set_rtl_option Tcl command to TRUE to enable GenSys add comments in the generated RTL.

For details on this argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, select the *Comments* checkbox in Figure 7-1.

Adding Parameter Declaration in the RTL

During RTL generation, GenSys provides you the following options related with generating parameter/generic values with instances in the RTL:

- Generating all parameter/generic values

  Set the value of the -params argument of the set_rtl_option Tcl command to all.

- Generating parameter/generic values different than the default values that is specified in the master component.

  Set the value of the -params argument of the set_rtl_option Tcl command to non_default.

For detailed information on the -params argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, select the *All* or *Non Default* checkbox in Figure 7-1.

Adding Connections Description in the RTL

To add a description of connections in the generated RTL, set the -print_conn_comment argument of the set_rtl_option Tcl command to TRUE.

For detailed information on the -print_conn_comment argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, select the *Connection Descriptions* checkbox in Figure 7-1.

Adding the Delta Delay Information in the RTL

To add delay information of connections in the generated RTL, set the -delta_delay argument of the set_rtl_option Tcl command to TRUE.

For detailed information on the -delta_delay argument, refer to the *GenSys Tcl Commands Reference Guide*.

The following example shows the delay information in the generated VHDL:

```
P1_delta_1 <= P1_INT(4 downto 2) after 5ns;P1_delta <= P1_INT(10 downto
8) after 2ns;
```

The following example shows the delay information in the generated Verilog:

```
assign #5 P1_delta_1 = P1[4:2];assign #2 P1_delta = P1[10:8];
```

Adding Header in the RTL

Before adding a header in the RTL, create The Header Template File. GenSys runs this template file and uses the generated output of this template file as the header of the generated RTL.

You can specify the template file by using the -header_template *<file-name>* argument of the set_rtl_option Tcl command.

For detailed information on the -header_template argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click

in Figure 7-1 and select the header template file.

The Header Template File

While creating the header file, you can make use of the name of the active RTL file and the active language. The name of the active RTL file and the active language are returned by the getActiveFile and getActiveLang Perl APIs, respectively.

The sample header template file is as follows:

```
----------------------------------------------------
--H
[%-SET file = root.active("activefile")-%]
[%-SET lang = root.active("activelang")-%]
--H Filename : [%file%]
--H Language : [%lang%]
---------------------------------------------------
```

Adding Verilog Timescale Directives in the RTL

Some simulators need Verilog timescale directives in the generated Verilog.

To generate these directives in the generated Verilog, set the -dump_timescale argument of the set_rtl_option Tcl command.

For detailed information on the -dump_timescale argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click the *Verilog* button in The Set RTL Options Dialog. The *Verilog Generation Options* dialog appear in which select the *Include Timescale Directive* checkbox.

Adding Component Declaration in the RTL (VHDL)

To add component declaration in the architecture of the parent module in the generated VHDL, set the -component_decl argument of the set_rtl_option Tcl command.

For detailed information on the -component_decl argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click the *VHDL* button in Figure 7-1. The *VHDL Generation Options* dialog appear in which select the *Include Comp. Declarations in Arch. Def.* checkbox.

Generating Optimized Nets in the RTL

To generate optimized nets in the generated RTL, set the -optimized_nets argument of the set_rtl_option Tcl command to TRUE.

For detailed information on the -optimized_nets argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click the *Advanced* button in Figure 7-1. The *Advanced RTL Options* dialog appear in which select the *Optimize Wire Creation* checkbox.

The following figure shows the difference between the optimized and unoptimized nets:

Optimized nets

```
module TEST_TOP ();
wire [1:0] inst1_R1;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_0;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_2;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_0;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_2;
comp1 inst1 (
.R1({UNINTENTIONAL_OPEN_inst1_R1_2,
inst1_R1,
UNINTENTIONAL_OPEN_inst1_R1_0}) //O:6
);
comp2 inst2 (
.R2({UNINTENTIONAL_OPEN_inst2_R2_2,
inst1_R1,
UNINTENTIONAL_OPEN_inst2_R2_0}) //I:6
);
endmodule
```

Unoptimized nets

```
module TEST_TOP ();
wire [5:0] inst1_R1;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_0;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_2;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_0;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_2;
comp1 inst1 (
.R1({UNINTENTIONAL_OPEN_inst1_R1_2,
inst1_R1[3:2],
UNINTENTIONAL_OPEN_inst1_R1_0}) //O:6
);
comp2 inst2 (
.R2({UNINTENTIONAL_OPEN_inst2_R2_2,
inst1_R1[3:2],
UNINTENTIONAL_OPEN_inst2_R2_0}) //I:6
);
endmodule
```

Handling Holes in the Generated RTL

Holes refer to the unconnected bits of ports that are generated on the module boundary during grouping or partitioning.

To allow or suppress the existence of holes in the generated RTL, use the -holes_in_rtl_hierarchy <*TRUE* | *FALSE*> argument of the set_rtl_option Tcl command.

For detailed information on the -holes_in_rtl_hierarchy argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click the *Advanced* button in Figure 7-1. The *Advanced RTL Options* dialog appear in which select the *Allow holes in hierarchy* checkbox.

Adding VSS and VCC Information in the Generated RTL

GenSys provides you the flexibility to decide if:

- VSS/VCC should be generated in place of 1'b0/1'b1 in the generated Verilog.
- VSS/VCC should be generated in place '0'/'1' in the generated VHDL.

Specify the required setting by using the -useSupplyName <*TRUE* | *FALSE*> argument of the set_rtl_option Tcl command.

You can also provide a supply name for the RTL by using the -rtl_supply0_name and -rtl_supply1_name argument of the set_rtl_option Tcl command.

For details on these arguments, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click the *Advanced* button in Figure 7-1. The *Advanced RTL Options* dialog appear in which specify the supply names in appropriate text boxes.

Adding Black Box Information in the Generated RTL

To add the empty architecture (black box) information in the generated RTL, set the -black_box_dump argument of the set_rtl_option Tcl command to TRUE.

For detailed information on the -black_box_dump argument, refer to the *GenSys Tcl Commands Reference Guide*.

Preserving MSB and LSB in the Generated RTL

To retain the MSB and LSB information of connections in the generated RTL, set the -preserve_lsb_msb argument of the set_rtl_option Tcl command to TRUE.

For detailed information on the -preserve_lsb_msb argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click the *Advanced* button in Figure 7-1. The *Advanced RTL Options* dialog appear in which select the *Preserve LSB/MSB of Connected Ports* checkbox.

Specifying the Format of VHDL Architectures in the Generated RTL

To specify the format of architecture names in the generated VHDL, use the -archName <*architecture-name-format*> argument of the set_rtl_option Tcl command.

For detailed information on the -archName <*architecture-name-format*> argument, refer to the *GenSys Tcl Commands Reference Guide*.

Specifying VHDL Generation Style

GenSys provides you the flexibility to generate separate or single VHDL files for entities and architectures of a module.

Use the -dumpVHDLSingleFile <*TRUE* | *FALSE*> argument of the set_rtl_option Tcl command to specify the VHDL generation style.

For detailed information on the -dumpVHDLSingleFile <*TRUE* | *FALSE*> argument, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, click the *VHDL* button in Figure 7-1. The *VHDL Generation Options* dialog appear in which select or deselect the *Create Single VHDL File Per Module* checkbox.

Customizing Net Names in the Generated RTL

Use the following arguments of the set_rtl_option Tcl command to add customized net names in the generated RTL:

- -default_tieoff_prefix <*default-prefix*>
- -forced_tieoff_prefix <*forced-prefix*>
- -temp_tieoff_prefix <*temp-prefix*>

For details on these commands, refer to the *GenSys Tcl Commands Reference Guide*.

Alternatively, use Perl to customize net names. For details, refer to the *Creating a Generator for Changing Net Names in Generated RTL* section in *GenSys Customization Guide*.

Using the spyglass.f File

Use this file as an input to the *SpyGlass* tool to run various checks on the generated RTL.

This file contains the following information:

- -mixed option as both VHDL and Verilog could be present.
- -sort option to ensure compilation order
- Logical to actual library mapping that is generated during RTL import.

  For example, -lib L1 L2. In this example, L1 is logical library and L2 is physical library.
- Logical name information associated with files in file sets.

  For example, -libhdlfiles L1. This specifies a list of VHDL files which are compiled to L1.
- -f sources.f option to invoke sources.f file.

Following is a sample spyglass.f file:

```
-mixed \
-sort \
-lib Logical1 Physical1 \
-libhdlfiles Logical1 '/soc/mux1.vhd /soc/adder1.vhd
/soc/mult1.vhd' \
-lib Logical2 Physical2 \
-libhdlfiles Logical2 '/soc/mux2.vhd /soc/
```

adder2.vhd /soc/mult2.vhd' \
-lib Logical3 Physical3 \
-libhdlfiles Logical3 mux3.vhd
-lib Logical4 Physical4 \
-lib Logical5 Physical5 \
-lib Logical6 Physical6 \
-f sources.f

Using the sources.f File

This file contains complete information about RTL files generated in GenSys as well as glue files generated during RTL import. This file is invoked from the spyglass.f file.

This file contains the following information:

- List of glue files under the Glue directory
- List of package files
- List of Verilog/VHDL files

Following is a sample sources.f file:

```
/soc/pack1.vhd \/soc/pack2.vhd \/soc/pack3.vhd \/vhdl/glue1.vhd \/vhdl/
glue2.vhd \/vhdl/glue3.vhd \/vhdl/leaf1.vhd \/vhdl/leaf2.vhd \/vhdl/
top.vhd
```

Selective RTL Generation

This section covers the following topics:

- Selective RTL Generation of RTL Modules and Entities
- Selective RTL Generation Internal to Modules and Entities

Selective RTL Generation of RTL Modules and Entities

During RTL generation, GenSys selectively generates RTL for only those modules that are either created or modified in GenSys.

The *sources.f* file contains information of GenSys generated modules and unchanged modules that were imported during RTL import. See Modules Eligible for Selective RTL Generation.

Consider the scenario in which you import the des1, des2, des3, and topDes modules where des1, des2, and des3, are instantiated in topDes.

Now consider that you modify the connectivity of the des1 and des3 modules. In this case, when you generate RTL, the *sources.f* file will contain GenSys-generated des1 and des3 modules and original des2 and topDes modules.

Note:
This is the default flow in GenSys.

Modules Eligible for Selective RTL Generation

Modules that are eligible for Selective RTL Generation are the designs/components that have been modified with respect to the input RTL.

In case of a flat file, that is, a file having more than one module/entity-architecture definition, GenSys generates the edited module in line.

For example, if a file contains low, mid, and top modules and mid has been modified, GenSys generates a flat RTL only with RTL description of mid changed. This way, the generated RTL looks very close to the input RTL with only incremental changes.

Selective RTL Generation Internal to Modules and Entities

The *Selective RTL Generation Internal to Modules and Entities* flow preserves the look-and-feel of the generated RTL as close as possible to the initial RTL.

In the RTL restructuring flow:

- If the user does not edit the unsynthesizable constructs within GenSys, this feature takes care of preserving such constructs.
- If the user just moves the instances across hierarchies, this feature preserves the pre-processor directives in the generated RTL.

  Note:
  This is the beta feature of RTL Assembly and should be used with the RTL restructuring flow only.

  Note:
  This feature does not work with the feature of Solidifying Glue Instances and Generate Blocks.

Enabling the Selective RTL Generation Flow

To enable this flow, perform the following actions:

- Use the gen_rtl_snl license.
- Set the -selective_netlisting argument of the set_gensys_options tcl command to Y.

Example of Selective RTL Generation

This section shows the following files in this flow:

- Original RTL Used as the Input
- RTL Generated when Selective RTL Generation is Disabled
- RTL Generated when Selective RTL Generation is Enabled
- Difference between Original RTL and Generated RTL when Selective RTL Generation is Disabled
- Difference between Original RTL and Generated RTL when Selective RTL Generation is Enabled

Original RTL Used as the Input

The example of the input RTL is as follows:

```
module des ();wire  [11:0] LPM_MFP0_BOOTSTRAP;  core_dsgn  core (
.MFP0_BOOTSTRAP(LPM_MFP0_BOOTSTRAP)  //I:12                      );
MFP    MFP0 (                  .BOOTSTRAP({
LPM_MFP0_BOOTSTRAP})   //O:12                );endmodule
```

RTL Generated when Selective RTL Generation is Disabled

Following is the RTL generated when selective RTL generation is disabled:

```
module des ();// MFP0 wires wire    [11:0]   LPM_MFP0_BOOTSTRAP;
core_dsgn    core ( //Instance       .MFP0_BOOTSTRAP(LPM_MFP0_BOOTSTRAP)/
/O:12   //In[11:0]                  );    MFP    MFP0 ( //Instance
.BOOTSTRAP(LPM_MFP0_BOOTSTRAP)//O:12   //Out[11:0]                );
comp1    comp10 ( //Instance
.iA(LPM_MFP0_BOOTSTRAP[8:0]),  //In[8:0]
.iB(LPM_MFP0_BOOTSTRAP[1])   //In[1]                    );endmodule
```

To view the difference between the input RTL and the above RTL, see Difference between Original RTL and Generated RTL when Selective RTL Generation is Disabled.

RTL Generated when Selective RTL Generation is Enabled

Following is the RTL generated when selective RTL generation is enabled:

```
module des ();wire  [11:0] LPM_MFP0_BOOTSTRAP;  core_dsgn  core (
.MFP0_BOOTSTRAP(LPM_MFP0_BOOTSTRAP)  //I:12                      );
MFP    MFP0 (                  .BOOTSTRAP({
LPM_MFP0_BOOTSTRAP})   //O:12                );comp1  comp10 (
.iA(LPM_MFP0_BOOTSTRAP[8 : 0]),    .iB(LPM_MFP0_BOOTSTRAP[1]));endmodule
```

To view the difference between the input RTL and the above RTL, see Difference between Original RTL and Generated RTL when Selective RTL Generation is Enabled.

Difference between Original RTL and Generated RTL when Selective RTL Generation is Disabled

The following figure shows the difference:

```
module des ();
wire    [11:0]   LPM_MFP0_BOOTSTRAP;

    core_dsgn    core (
                        .MFP0_BOOTSTRAP(
LPM_MFP0_BOOTSTRAP)   //I:12
                        );

    MFP    MFP0 (
                  .BOOTSTRAP({
                              LPM_MFP0_B
OOTSTRAP})   //O:12
                  );

endmodule
```

```
module des ();

// MFP0 wires
wire    [11:0]   LPM_MFP0_BOOTSTRAP;

    core_dsgn    core ( //Instance
                        .MFP0_BOOTSTRAP(LPM
_MFP0_BOOTSTRAP)//O:12   //In[11:0]
                        );

    MFP    MFP0 ( //Instance
                  .BOOTSTRAP(LPM_MFP0_BOOTS
TRAP)//O:12   //Out[11:0]
                  );

    comp1    comp10 ( //Instance
                      .iA(LPM_MFP0_BOOTSTRA
P[8:0]),  //In[8:0]
                      .iB(LPM_MFP0_BOOTSTRA
P[1])   //In[1]
                      );
endmodule
```

Difference between Original RTL and Generated RTL when Selective RTL Generation is Enabled

The following figure shows the difference:

```
+--  7 lines: module des ();----------
    MFP    MFP0 (
                  .BOOTSTRAP({
                                LPM_MFP0
_BOOTSTRAP})   //0:12
                );

endmodule
```

```
+ +--  7 lines: module des ();--------------
    MFP    MFP0 (
                  .BOOTSTRAP({
                                LPM_MFP0_BOO
TSTRAP})   //0:12
                );

comp1  comp10 (
    .iA(LPM_MFP0_BOOTSTRAP[8 : 0]),

    .iB(LPM_MFP0_BOOTSTRAP[1]));

endmodule
```

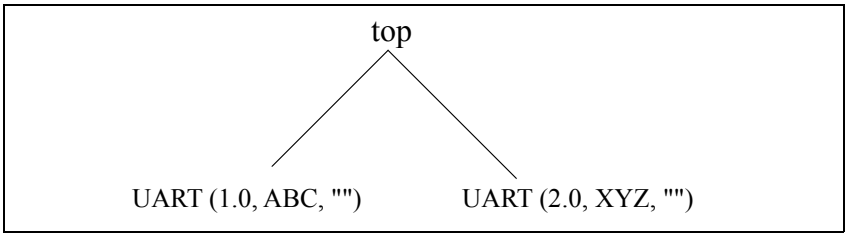
RTL Generated for the Objects Having Same Name

In an RTL, each primary design unit is uniquely identified by its name. Therefore, the generated RTL cannot have multiple primary design units of same name.

However, in GenSys, an object is uniquely identified by the combination of its VLNV (Vendor, Library, Name, and Version). Therefore, while working in GenSys, you can create/load multiple objects of same name but different VLV (Vendor, Library, and Version). For example, you may load two objects of the name, UART, which have different versions, 1.0 and 2.0.

In such cases, when you generate RTL, GenSys creates unique names (UART and UART_2_0) of such objects in the generated RTL.

For example, consider a design, top, that contains two instances of the same names, UART, as shown in the following figure:



In the above scenario, the VLV of one instance is (1.0, ABC,"") and for the other instance, it is (2.0, XYZ,"").

Now while generating RTL of instances in this case, GenSys first encounters the UART instance of VLV (1.0, ABC,"") and dumps the object name as UART. However, when GenSys encounters another instance of the same name, UART, it picks the version of this instance and dumps the object name as UART_2_0.

8

GenSys GUI Modes

You can work in any of the following views in GenSys:

- Canvas view

  In this view, you work directly through schematic. To switch to this view, click

  button present on the right side of GenSys work area.

  This is the default view when you load GenSys.

  Note:

  Currently, the canvas view is supported only for GenSys assembly, GenSys registers, and GenSys IO.

- Tabular view

  In this view, you work through tables. To switch to this view, click

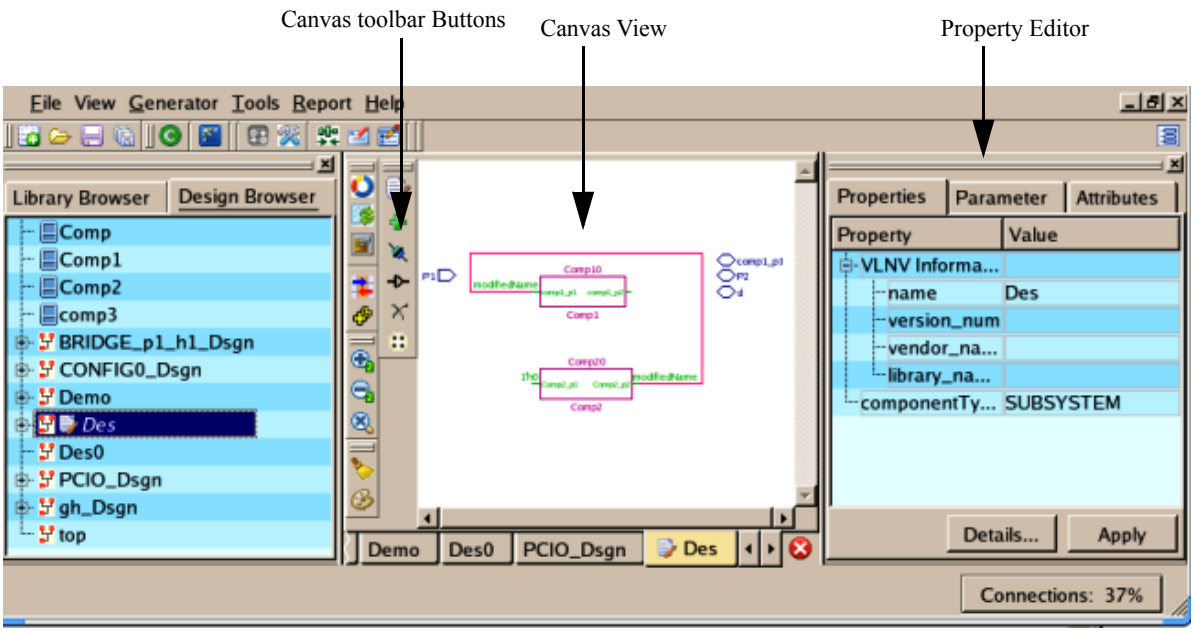
  button present on the right side of GenSys work area.

When you close GenSys session, the view at that time is stored in .gensysrc file. So next time when you load GenSys, the view active in the last session is loaded.

Using the Canvas View

This view is schematic centric in which all the tasks are performed directly through the schematic.

The following figure shows the canvas view:



In the above view:

- Use the toolbar buttons to perform actions, such as creating objects, loading instances, and adding ports.

  For details, see Performing Various Actions in the Canvas View.

- Use the property editor to add or modify details, such as object properties, parameters, and attributes of the currently selected object in the schematic.

  For details, see Using the Property Editor Pane.

When you close the canvas view, position of various dialogs are saved in the .gensysrc file. Next time when you switch back to this view, all dialogs are displayed in the same position.

Performing Various Actions in the Canvas View

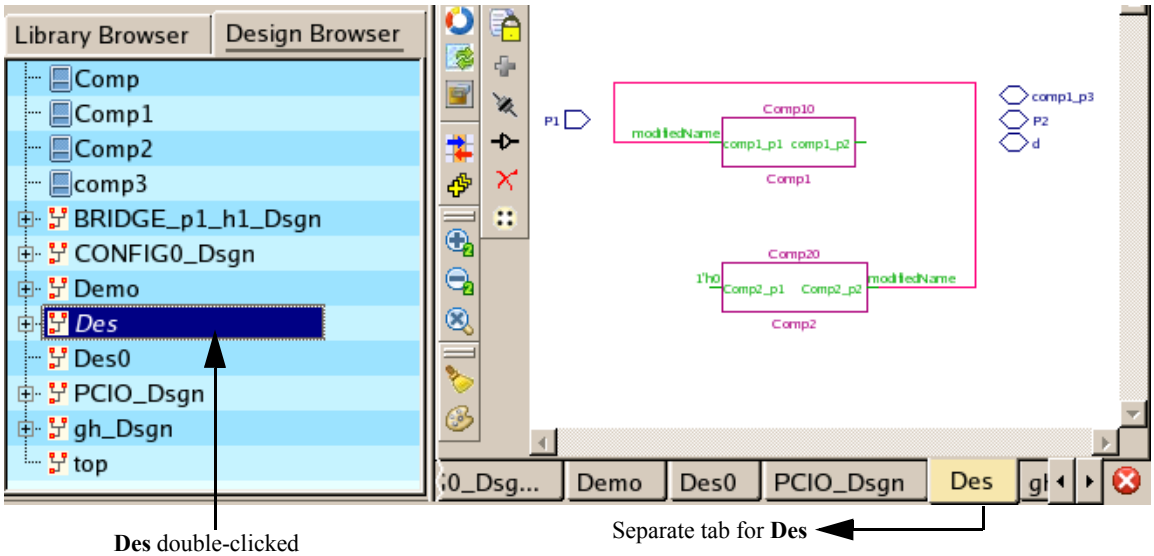
This section describes actions that you can perform in the canvas view.

Entering into the Edit Mode

Before modifying an object, enter into the edit mode by performing the following steps:

1. Double-click on that object appearing under the Design Browser Tab.

   A tab for that object appears under the canvas view, as shown in the following figure:



2. Click

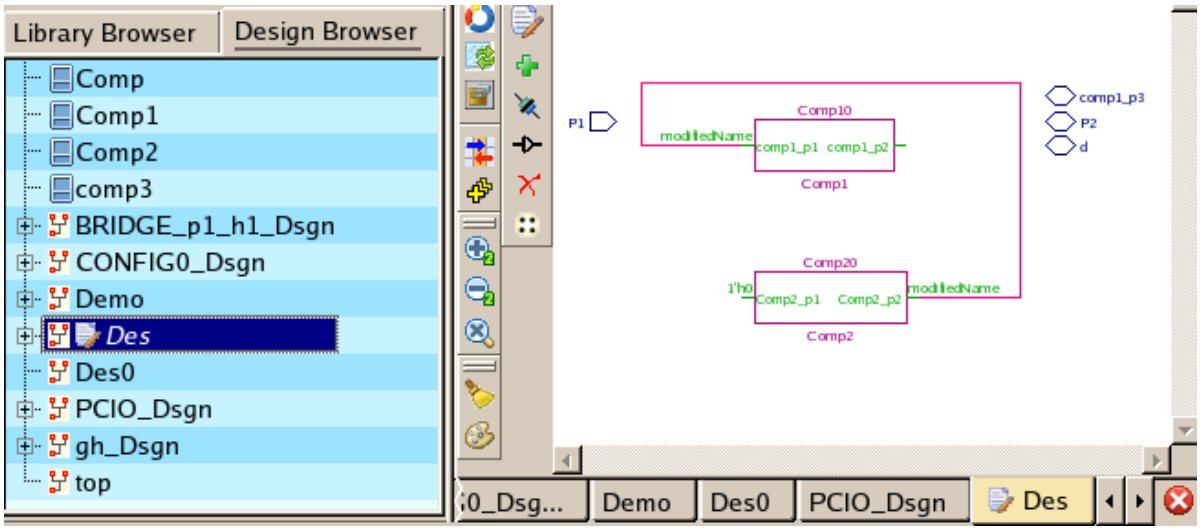
3. in the canvas toolbar to enter into the edit mode for the selected object.

   When you click this button, it changes to

   indicating that the selected object is now editable. In addition, the

   icon appears before the name of the object under the Design Browser Tab and the tab highlighted under the canvas view.

   This is shown in the following figure:



At a time, only one object is editable.

For example, in the above figure, if you double-click on the Demo design under the Design Browser Tab, GenSys will open that design in the read-only mode.

When you change the mode of the Demo design to the edit mode, the Des design enters into the read mode.

Loading Objects from Files

To load objects (design, component, or interface) from files, perform the following steps:

1. Select the File > Open menu option.

   The Select one or more files to open dialog appears.

2. From this dialog, select the objects to be loaded.

Alternatively, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click in the canvas view and select the Open Existing Design/Component/Interface option from the shortcut menu.

   The Select one or more files to open dialog appears.

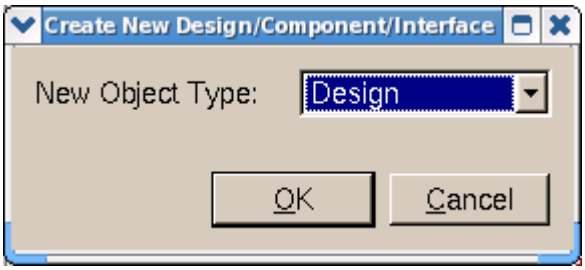
3. From the above dialog, select the objects to be loaded.

Creating New Objects

To create new objects, such as design, component, or interface in the canvas view, perform the following steps:

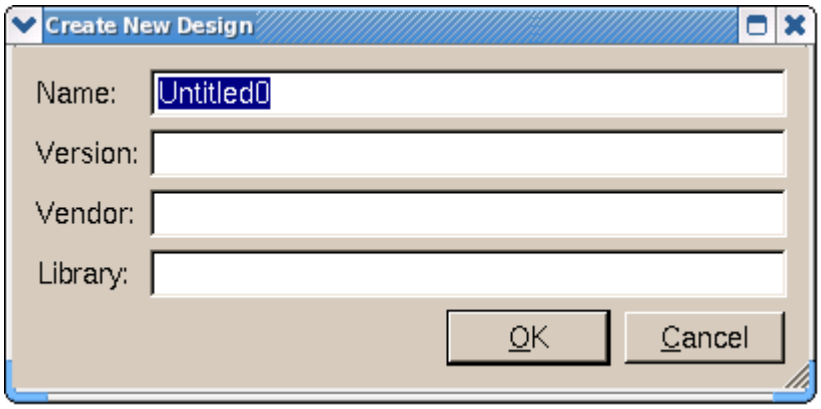
1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click in the canvas view and select the Create New Design/Component/Interface option from the shortcut menu.

   The Create New Design/Component/Interface dialog appears, as shown in the following figure:



3. Select the type of the object to be created from the New Object Type drop-down list in the above dialog.
4. Click the OK button to close the above dialog.

   The Create New <object-type> dialog appears based on the type of object. For example, the following figure shows the Create New Design dialog:



5. Specify the required details, such as name, version, vendor, and library in the above dialog.

6. Click the OK button to close the above dialog.

After performing the above steps, the specified object is created.

Deleting Objects

To delete any object, instance, or connection, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Select an object/connection, and click the

3. button in the canvas toolbar.

You can select multiple objects, instances, or connections by keeping the *Shift* key pressed.

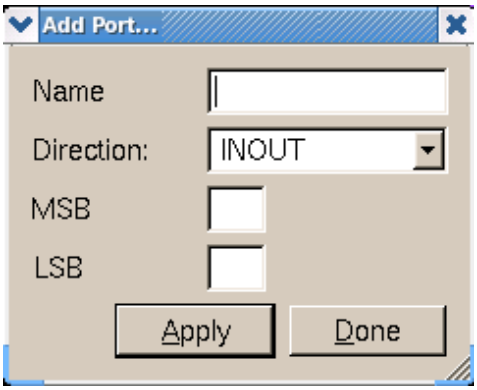
Adding Ports

To add a port, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Click the

3. button in the canvas toolbar.

   This displays the Add Port dialog, as shown in the following figure:



4. In the above dialog, specify the required details for the port to be added.

5. Click the Done button.

After performing the above steps, the specified port gets added in the design.

Exporting Pins an Interfaces to the Design Boundary

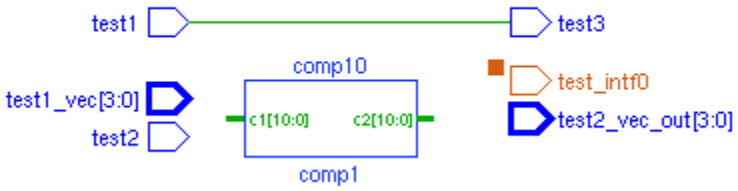
Exporting an instance pin or an interface means making the interface of an IP (instantiated in the design) or the pin of an instance in the design visible at the top-level of the design.

After the export process, an adhoc connection appears between the exported pin on the design and the instance pin or the exported interface on the design and the interface on the IP instance in the design.

Exporting Pins on the Design Boundary

Consider the following schematic of a design:

Figure 8-1 Exporting Pins to the Design Boundary



Now consider that you want to export the c1 pin of the comp10 instance. To do so, perform the following steps:

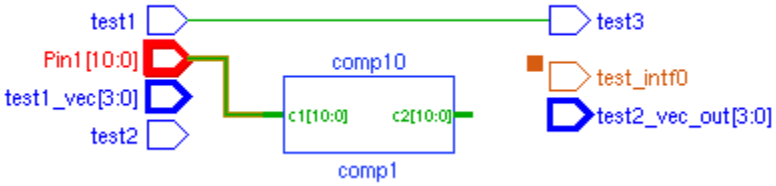
1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click on the c1 pin of the comp10 instance.
3. Select the Export Pin/Interface option from the shortcut menu. The Export Pin dialog appears.
4. Specify the required details in the Export Pin dialog, as shown in the following figure::

In the above dialog, you can specify the following details:

- Name of the pin to be exported in the Export Name text box. This text box is disabled when multiple pins are selected.
- Prefix and Postfix specify the prefix and postfix that needs to be appended to the name after it is exported.
- MSB and LSB of the port to be created on the design using the MSB and LSB drop-down boxes, respectively, under the Top Design section.
- MSB and LSB of the pin by using the MSB and LSB drop-down boxes, respectively, under the Instance section.

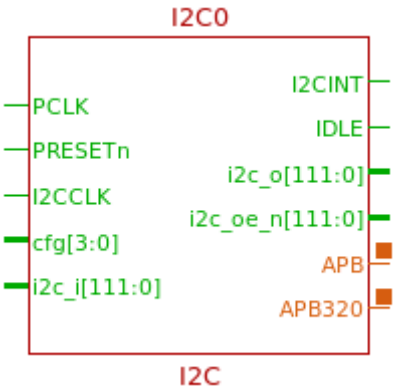
5. Click the OK button.

After performing the above steps, the c1 pin of the comp10 component is exported to the design as the Pin1 pin, and an adhoc connection is created between c1 and Pin1. So Figure 8-1 now appears as the following:



Exporting Interfaces on the Design Boundary

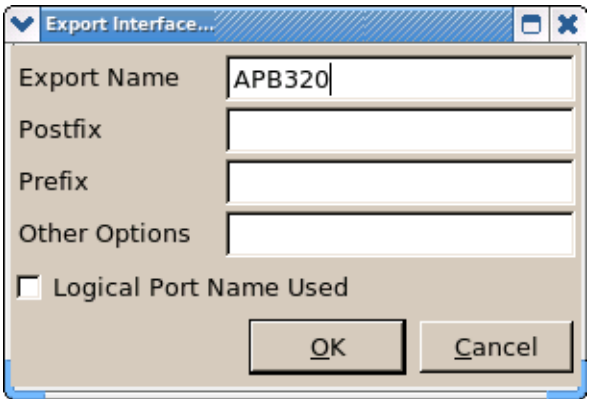
Consider the following schematic of a design containing the I2C0 instance:



Now consider that you want to export the APB230 interface of the I2C0 instance. To do so, perform the following steps:

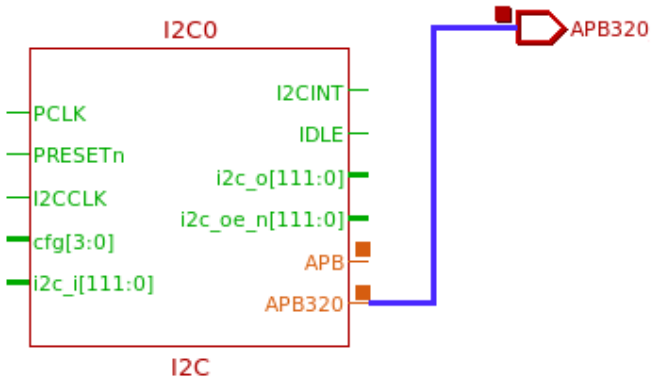
1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click on the APB230 interface.
3. Select the Export Pin/Interface option from the shortcut menu.

   The Export Interface dialog appears, as shown in the following figure:



4. Specify details in the Export Interface dialog.
5. Click the *OK* button to close the Export Interface dialog.

After performing the above steps, the interface appears on the design boundary and an adhoc connection appears between the instance interface and the exported interface. This is shown in the following figure:



Creating Direct Connections Between Objects

A direct connection refers to an adhoc connection between ports/pins. You can create a direct connection in any of the following ways:

Method 1

In this method, perform the following steps to create a direct connection between pins/ports:

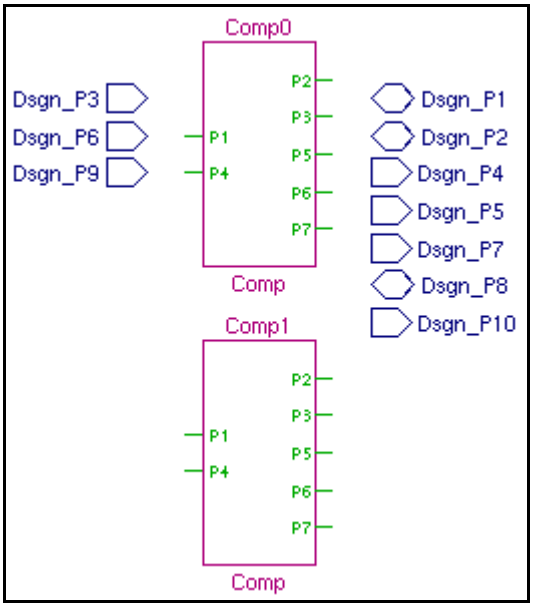
1. Enter into the edit mode. See Entering into the Edit Mode.
2. Select the pins/ports to be connected by keeping the *Shift* key pressed.
3. Right-click on the selected pins/ports, and select the Add Direct Connection option from the shortcut menu.

Method 2

In this method, direct connections are created when you click the

button in the schematic toolbar.

For example, consider the following schematic of a design:



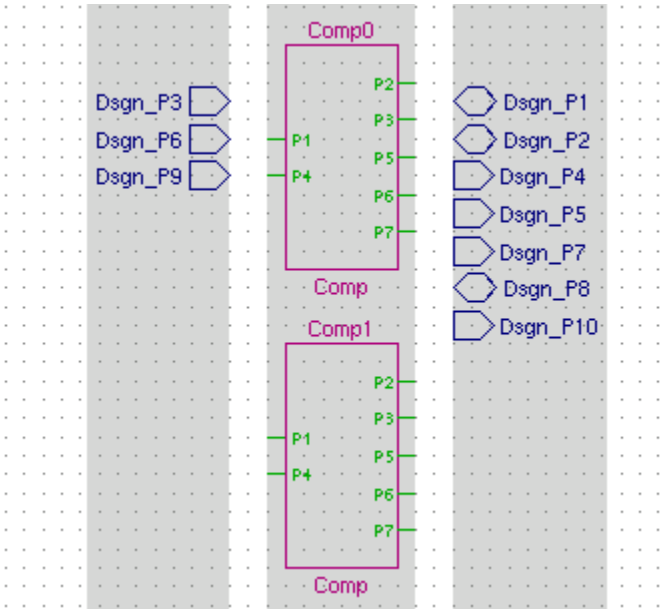
In the above schematic, consider that you want to create a direct connection between the P2 pin of the Comp0 instance and the Dsgn_P1, Dsgn_P2, and Dsgn_P4 pins of the design.

To create this connection, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Click the

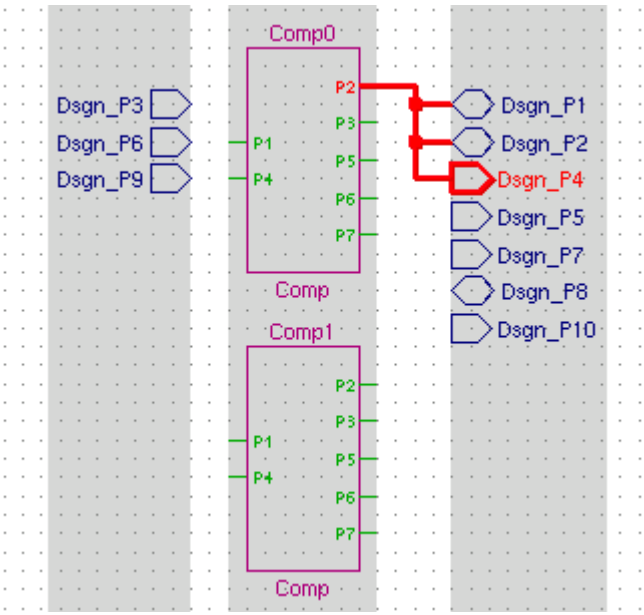
3. button in the canvas toolbar to enter the Connection Mode.

   The appearance of the schematic changes to the following:



4. Start selecting the pins to be connected by keeping the <*Shift*> keyboard key pressed.

   As you select the pins, a direct connection appears between the selected pins. This is shown in the following figure:



To exit the Connection Mode, press the Esc key or click the

button in the schematic toolbar.

Creating Connections with Additional Details

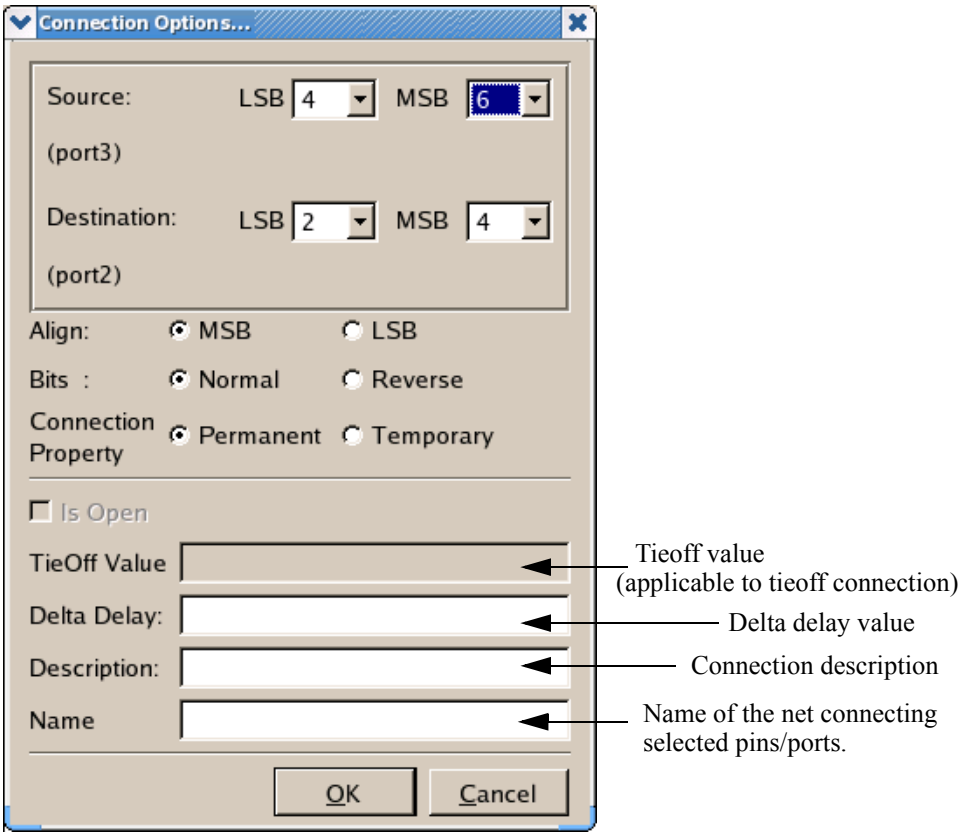
While creating connections, you can specify details, such as MSB, LSB, and normal/reversed bits. You can create such connections in any of the following ways.

Method 1

In this method, perform the following steps to create a connection between pins/ports:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Select the pins/ports to be connected by keeping the *Shift* key pressed.
3. Right-click on the selected pins, and select the Add Connection option from the shortcut menu.

   The Connection Options dialog appear, as shown in the following figure:



4. In the above dialog, specify the required details.
5. Click the OK button.

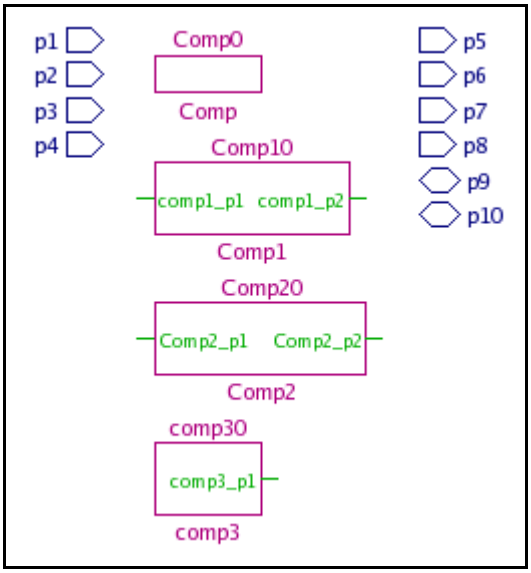
After performing the above steps, a connection gets created between the selected pins/ ports.

Method 2

In this method, connections are created when you click the

button in the schematic toolbar.

For example, consider the following schematic of a design:



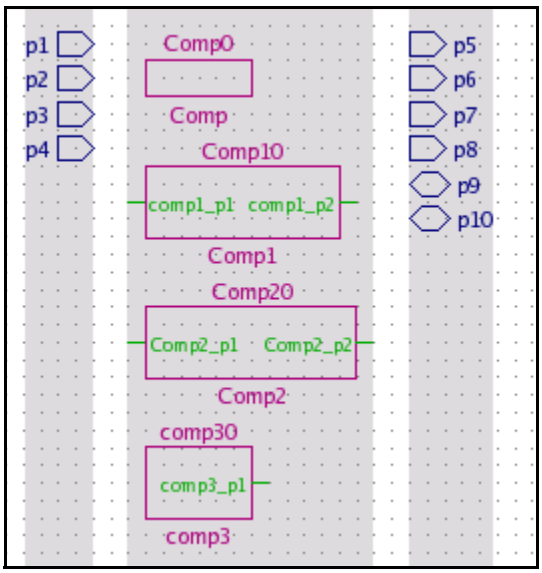
In the above schematic, consider that you want to create a temporary connection between the comp1_p1 pin of the Comp10 instance and the P5 port of the design, such that the connection is MSB-aligned.

To create this connection, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Click the

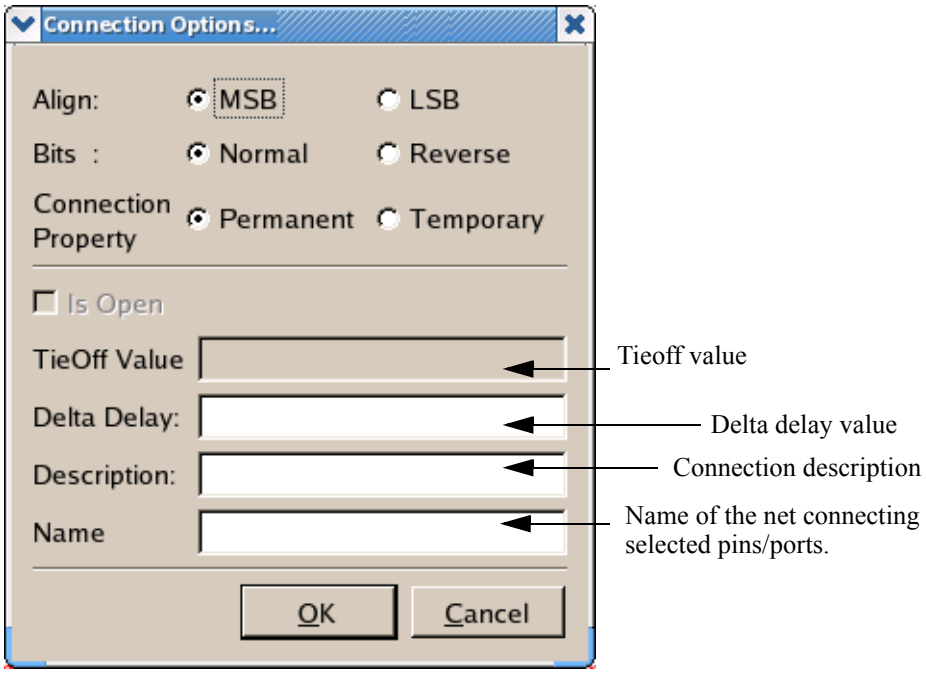
3. button in the canvas toolbar to enter in the connection mode.

   The appearance of the schematic changes to the following:



4. Select the comp1_p1 pin of the Comp10 instance and the P5 port of the design by keeping the <*Shift*> keyboard key pressed.

   When you select these pins, the Connection Options dialog appears, as shown in the following figure:



5. In the above dialog, specify the required details.
6. Click the OK button.

After performing the above steps, a connection gets created between the selected pins/ ports.

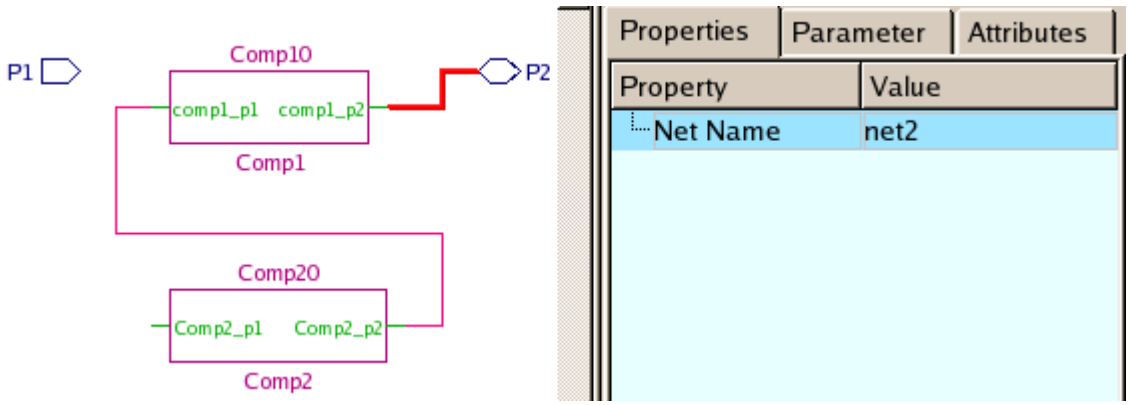
To exit from the connection mode, press the *Esc* key or click the

button in the schematic toolbar.

Specifying a Net Name for a Connection

When you create a connection, GenSys assigns a default name to the net created for that connection.

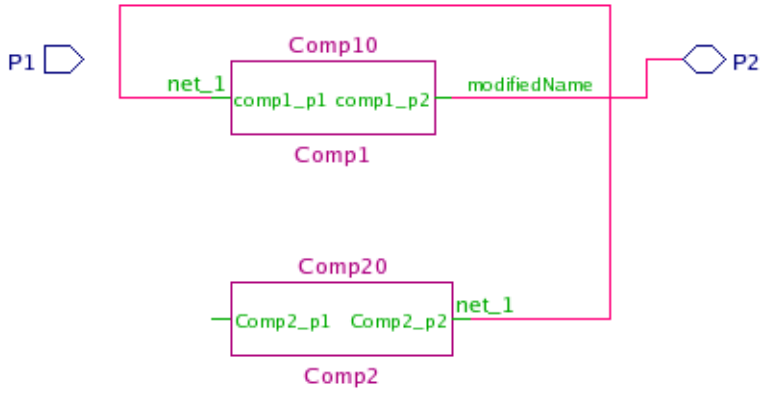
When you select that net, its name appears in the Property Editor pane, as shown in the following figure:



To change the net name, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Select a net in the canvas view.
3. Change the net name of the NetName property displayed under the Properties tab of the Property Editor pane.
4. Click the Apply button in the

After performing the above steps, the net name appears for the net, as shown in the following figure:



Note:
For net names to be visible in the schematic, Ensure that the Show Net Names in Schematic check box is selected in the Schematic Options.Misc tab of the Options dialog.

Note:
GenSys does not allow you to change the names of read-only nets and nets that appear dotted in the schematic.

Deleting Connected Nets

To delete a net connecting two ports, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Select the connected ports.

   You can select the connected ports by keeping the *Shift* key pressed on the keyboard.
3. Right-click on any selected port, and select the Delete Connected Net option from the shortcut menu.

After performing the above steps, the net is deleted between the selected ports.

Adding Glue Logic in a Design

Glue refers to the extra logic that exists in a design apart from instantiations of components and sub-systems.

You can insert any of the following types of glue logic in a design:

- A pre-defined type of glue logic, such as AND, NAND, OR, NOR, NOT, XOR, XNOR, FLOP, BUF, or MUX (2X1)

For details, see Inserting a Predefined Glue Logic.

- A customized glue logic

  For details, see Inserting a Customized Glue Logic.

Such glue logic is also known as a pseudo component because such components are dissolved during RTL generation. However, the specified glue behavior remains in the parent RTL module. During RTL import, such logic is visible as a glue instance.

You can add ports and parameters for such glue logic and then create connections through these ports. For details on adding ports and parameters, see Adding Ports and Parameters for Glue Logic.

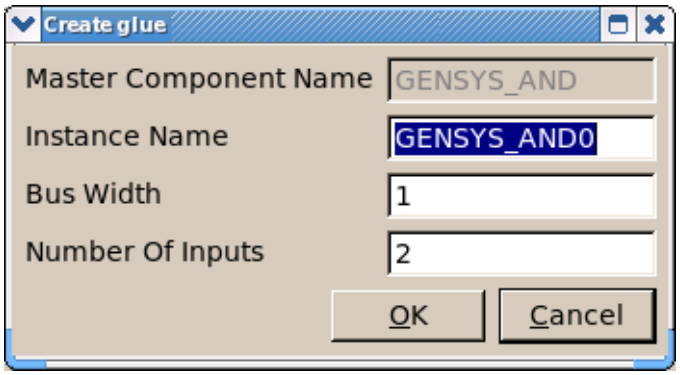
Inserting a Predefined Glue Logic

To add a predefined glue logic in your design, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click on the schematic of the design, and select the Create Glue Component option from the shortcut menu.

   A sub-menu appears.
3. Select the AND, NAND, OR, NOR, XOR (2X1), XNOR (2X1), MUX (2X1), FLOP, NOT, or BUF option from the sub-menu.

   The Create Glue dialog appears. For example, the following dialog appears when you select the AND option:



4. In the above dialog, specify the following details:
   - Name of the glue instance (Instance Name textbox)
   - Number of input pins of the glue instance (Number of Inputs textbox)

For N input pins, GenSys names the pins in the in[0 - N-1] format. For example, if the number of input pins is 4, the pin names are in0, in1, in2, and in3.

GenSys names the output pin as out0.

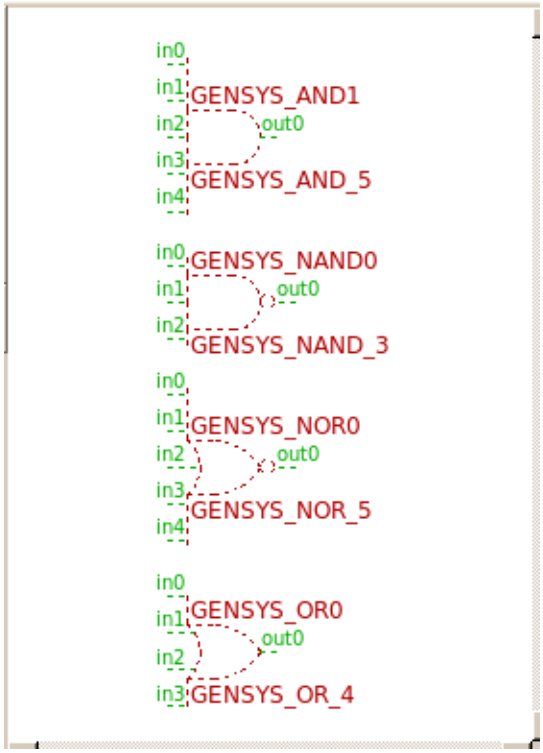
Note:

You can specify input pins only for the AND, NAND, OR, NOR, XOR, and XNOR logic. Therefore, the Number of Inputs textbox appears only for such logic.

- Number of bits for each input pin (Bus Width textbox)

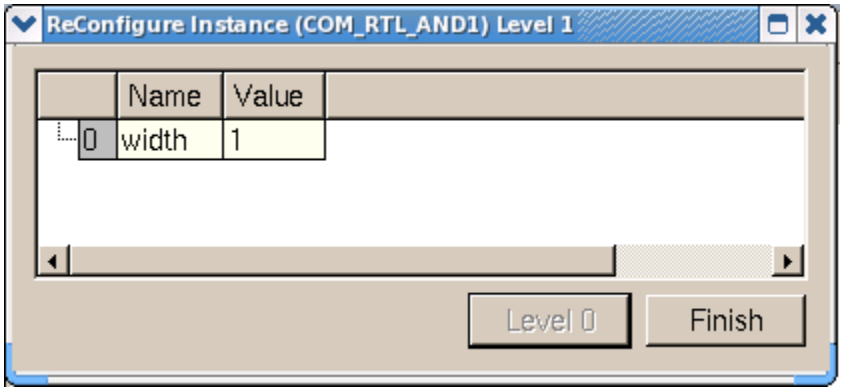
5. Click the OK button.

After performing the above steps, the specified glue logic is inserted in the design. The following figure shows some glue instances inserted in a design:



Configuring a Glue Logic

To configure a glue logic, right-click on the glue component and select the Configure Instance option from the shortcut menu. The Reconfigure Instance dialog appears, as shown in the following figure:



In the above dialog, specify the width of pins of the glue component, and click the Finish button.

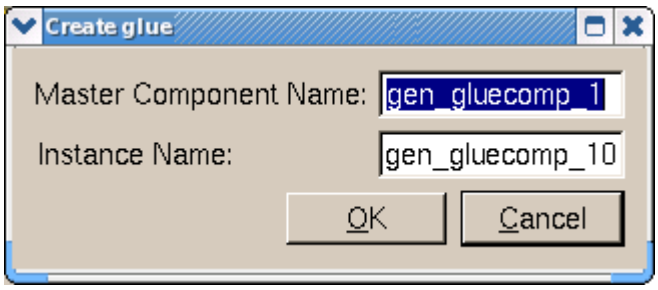
Inserting a Customized Glue Logic

To insert a customized glue logic, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click on the schematic, and select the Create Glue Component option from the shortcut menu.

   A sub-menu appears.
3. Select Custom from the sub-menu.

   The Component Name dialog appears. For example, the following dialog appears when you select the AND option:



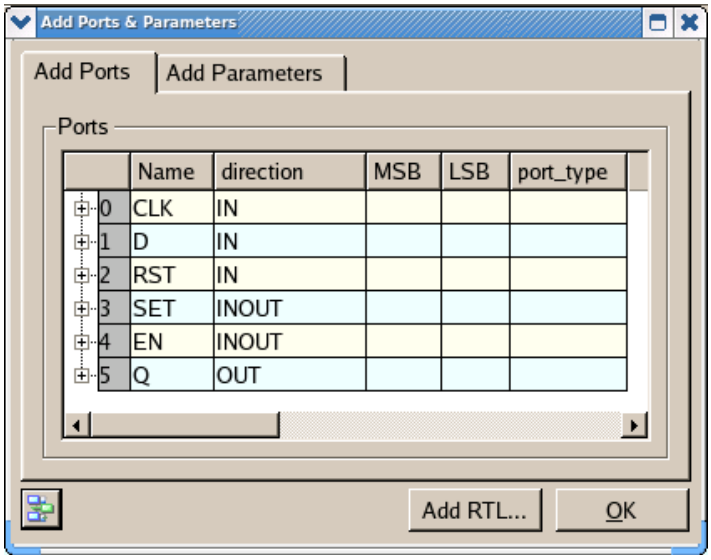
4. Specify the name of the master component and glue instance in the above dialog.
5. Click the OK button.

The Add Ports & Parameters dialog appears.

6. Specify the ports and parameters details in this dialog. For details, see Adding Ports and Parameters for Glue Logic.

Adding Ports and Parameters for Glue Logic

The following figure shows the Add Ports & Parameters dialog:



To open the above dialog, right-click on the glue logic in the Canvas View, and select the Add Ports/Parameters option from the shortcut menu.

In this dialog, perform the following actions:

- Add rows to specify ports and parameters information under the Ports and Parameters tab, respectively.
- Add RTL code by clicking the Add RTL button. This displays the Add RTL dialog in which you can make the required RTL updates.

Inferring Interfaces Based on Suggestions from the Tool

GenSys can suggest interfaces that you can infer on components and instances in a design. For each suggested interface, GenSys lists actual ports that should be the part of that interface.

To view and infer interfaces suggested by GenSys, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.

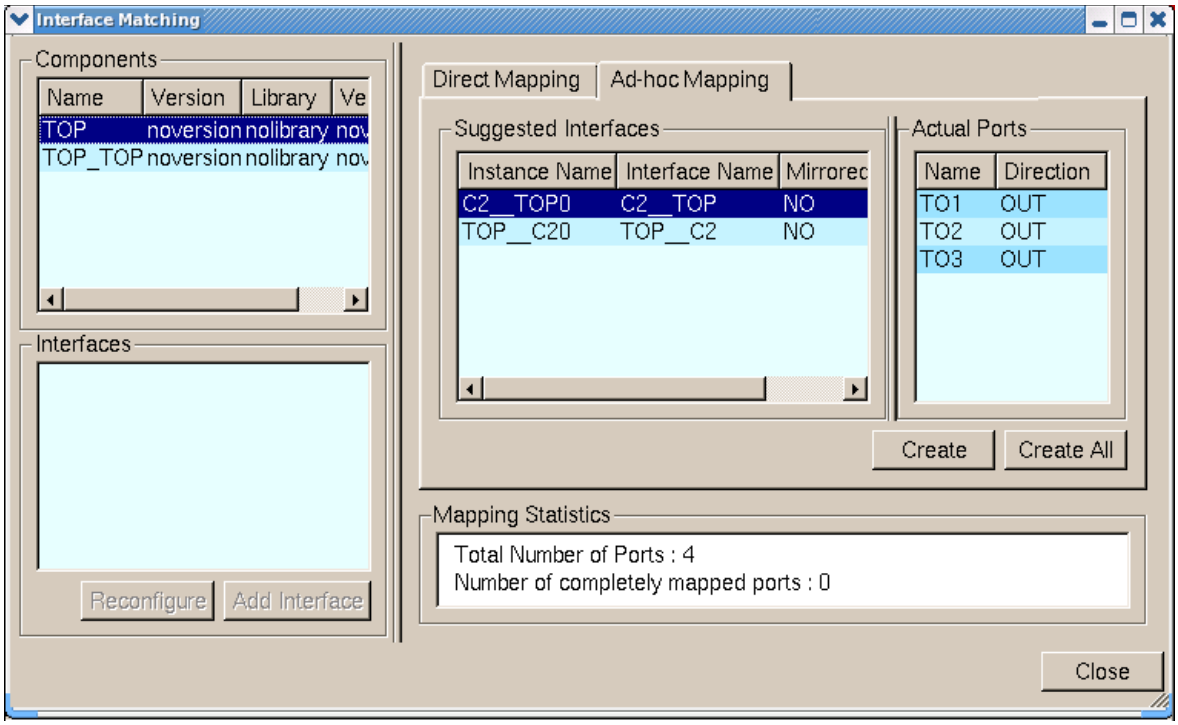
2. Right-click in the canvas view, and select the Map Interfaces option from the shortcut menu.

   The Interface Matching dialog appears.

3. In this dialog, select a component for which you want to view interface suggestions.

   The interfaces suggested by GenSys appears in the Ad-hoc Mapping tab.

   For example, the following figure shows the suggested interfaces for the TOP component:



4. In the above dialog, check the suggested interfaces list in the Suggested Interfaces section. When you select a suggested interface, actual ports that should be a part of that interface appear in the Actual Ports section.

5. Select the required interface that you want to infer on the selected component.

6. Click the Create button.

   To infer all the suggested interfaces on the selected component, click the Create All button.

After performing the above steps, GenSys infers the selected interfaces on the master of the selected component and on all its instances in the design hierarchy.

Displaying the Master of an Instance

To view the master of an instance, perform any of the following actions:

- Right-click on that instance and select the Open Master option from the shortcut menu.
- Right-click on that instance appearing under the Design Browser Tab and select the Open Master option from the shortcut menu.

This displays the master of the selected instance in the canvas view.

Refreshing the Canvas View

You can modify a design or component while keeping the Modular Schematic window open.

Once you are done with the changes, click the

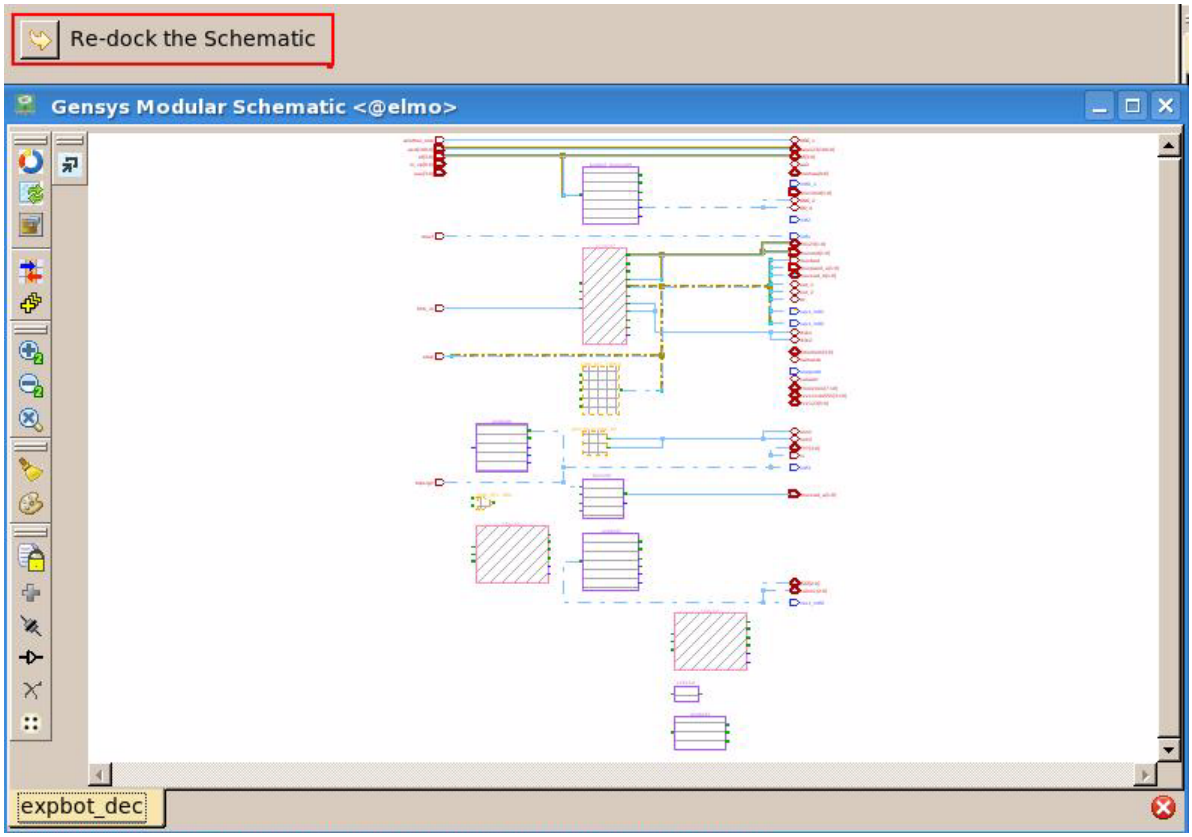
button in the canvas toolbar to reflect the changes in the schematic.

Undocking the Schematic

You undock the schematic from the Canvas View by clicking the

button in the canvas toolbar. This places the schematic in a new window.

To re-dock the schematic, click the Re-dock the Schematic button, as highlighted in the following image.

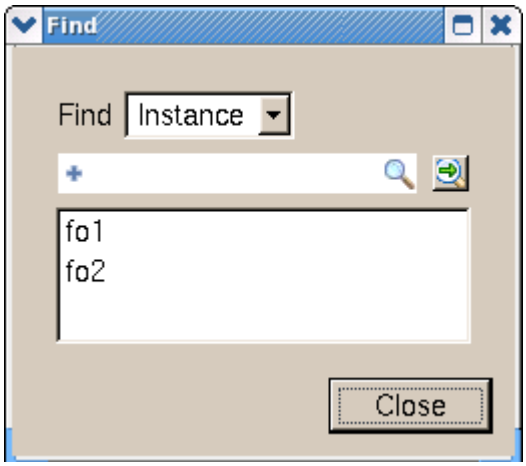


Searching Objects in the Schematic

To search various objects, such as instances, port, and pins in the schematic, click the

button in the canvas toolbar.

This displays the Find dialog, as shown in the following figure:



Use the above dialog to find various objects in the schematic by performing the following steps:

1. Select the type of object from the Find drop-down list.

   Names of all objects of the selected type are displayed.

2. Double-click on the name of the required object.

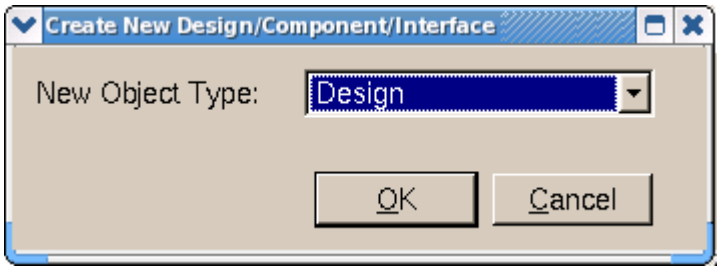
After performing the above steps, the view of the schematic is zoomed in to show the selected object.

Creating New Objects

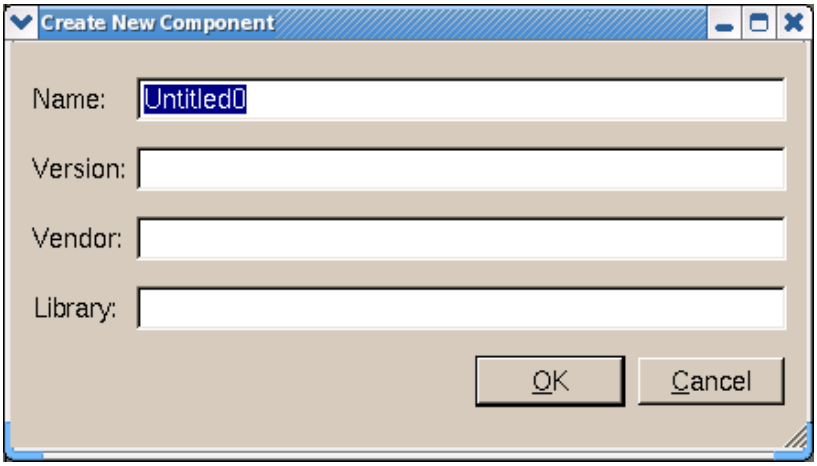
To create new objects directly through schematic, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click in the canvas view and select the Create New Design/Component/Interface option from the shortcut menu.

   The Create New Design/Component/Interface dialog appears, as shown in the following figure:



3. In the above dialog, select the type of object to be created from the New Object Type drop-down list.
4. Click the OK button.



This step displays the following dialog:

5. In the above dialog, specify the required details, such as name, vendor, version, and library of the object to be created.
6. Click the OK button.

After performing the above steps, the schematic of the previously loaded design is closed and a new schematic appears for the newly created object.

GenSys prompts you to save changes made in the previous schematic before opening the new schematic for the newly created object.

Modifying RTL of an Object

To view the RTL of an object, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Perform any of the following actions:
   - Right-click on an object in the schematic, and select the View RTL Source option from the shortcut menu.
   - Right-click an object of the active design under the Design Browser Tab, and select the View RTL Source option from the shortcut menu.

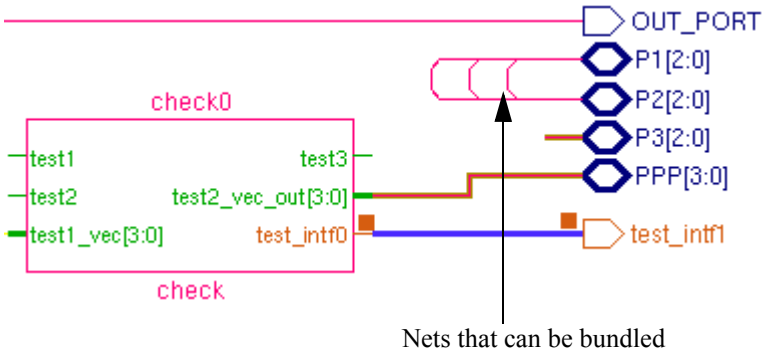
   The RTL of the object appears in a text editor window.

   - Update the RTL and save the changes.

Creating a Net Bundle

A net bundle is a collection of nets. It appears as a single thick net in a design.

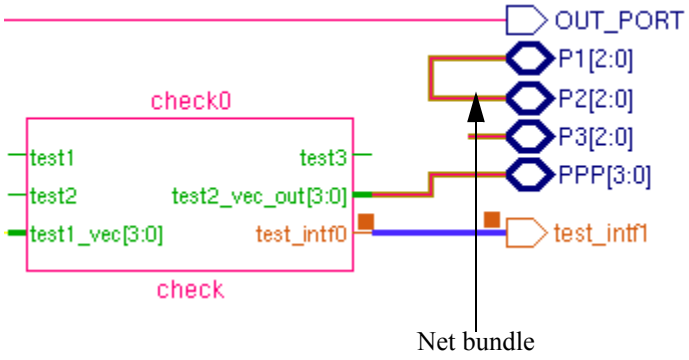
For example, consider the following schematic containing nets that can be bundled:



To bundle all such nets in the above schematic, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click in the canvas view and select the Bundle All Nets option from the shortcut menu.

The following figure shows the net bundle created for the above schematic:

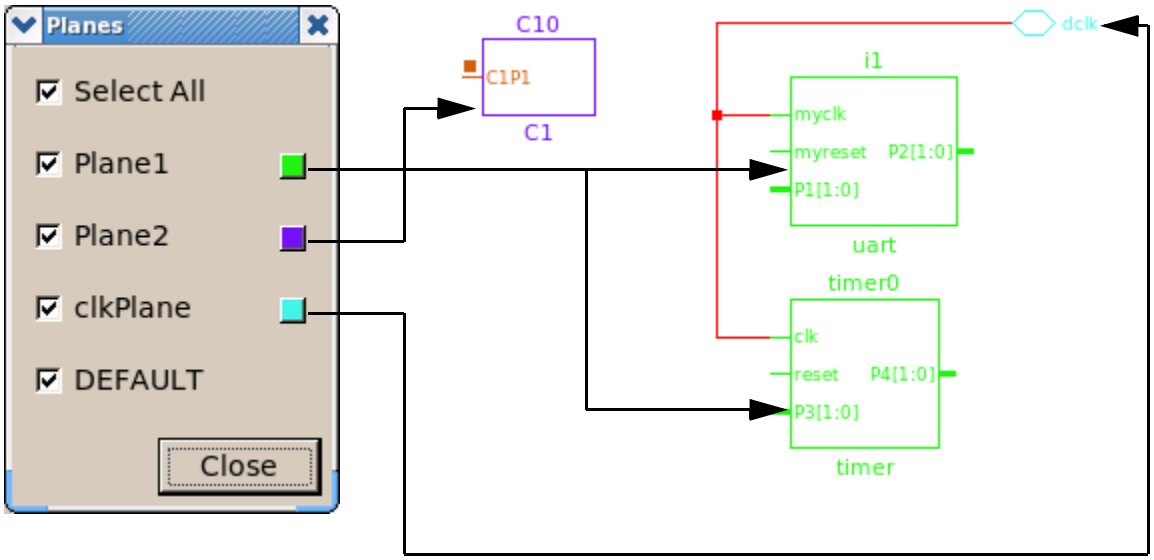


To un-bundle a net bundle, right-click on the net bundle and select the Unbundle option from the shortcut menu.

Working with Planes

A plane is a logical collection of objects, such as instance, ports, and components.

Each plane is assigned a unique color so that all the objects belonging to a plane appear in the plane color, as shown in the following figure:



An object belonging to multiple planes appear in the red color. For example, the connection between the myclk pin and the dclk port appears in the red color because the connecting objects belong to different planes (clkPlane and Plane1).

Note:
Connections do not have planes of their own. They are automatically assigned planes based on the pin/port to which they are connected.

By default, an object belongs to the DEFAULT plane. When you assign another plane to an object, the object is put on those planes. For details, see Assigning Planes to an Object.

This section covers the following topics:

- Showing and Hiding Planes
- Assigning Planes to an Object
- Changing the Plane Color
- Deleting a Plane

Showing and Hiding Planes

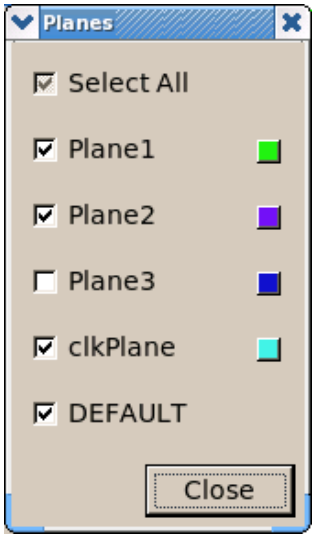
When you show or hide a plane, all the objects assigned to that plane appear or hide, respectively, in the schematic.

To show or hide planes, perform the following steps:

1. Click the

2. icon from the schematic toolbar.

   The Planes dialog appears, as shown in the following figure:

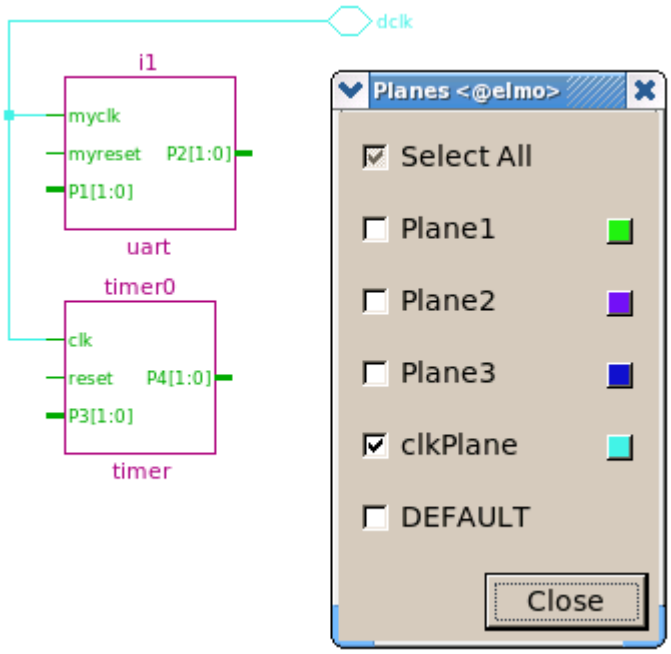


3. In the above dialog, select and de-select the planes that you want to show and hide, respectively.

   If you select or de-select the Select All option, all the currently loaded objects appear or hide, respectively, in the schematic.

4. Click the Close button.

Note that in some cases, an object appears even if its plane is de-selected in the Planes dialog. Such cases occur if that object is connected to some other object whose plane is not hidden. This is shown in the following figure:



In the above figure, the i1 and timer0 instances belong to the Plane1 plane, which is hidden. However, these instances appear in the schematic because they are connected to the dclk port that belongs to the clkPlane, which is not hidden.

Note:
An instance inherits planes from its master (component/design) and a pin inherits planes from the instance. The Planes dialog, however, does not show the inherited planes.

Assigning Planes to an Object

You can assign planes to an object in the following ways:

- Assigning Existing Planes to an Object

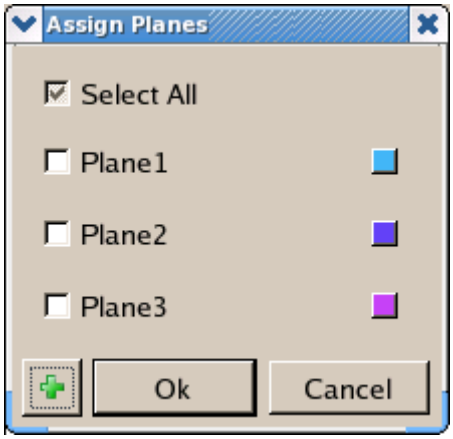
- Assigning Planes to the Master of an Object
- Creating Planes and Assigning Them to Objects
- Using Regular Expressions to Assign Planes

Assigning Existing Planes to an Object

To assign an existing plane to an object, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click on an instance in the schematic and select the Assign Planes option from the shortcut menu.

   The Assign Planes dialog appears, as shown in the following figure:



   Here, Plane1, Plane2, and Plane3 are already created before.

   Note:
   In the above dialog, only those planes are selected that are already assigned to the selected object. In this case, since no plane is assigned to the selected object, therefore no plane is selected.

3. From the above dialog, select a plane that you want to assign to the selected object.

   Note:
   You can also select multiple planes to assign multiple planes to an object.

4. Click the OK button.

After performing the above steps, the planes are assigned to the selected object.

Assigning Planes to the Master of an Object

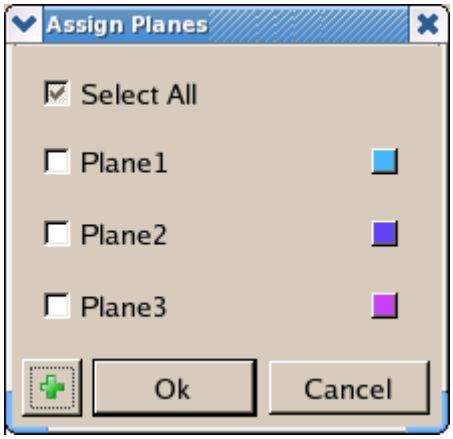
To assign planes to a master, right-click on an instance and select the Assign Planes to Master option from the shortcut menu. This opens the Assign Planes dialog using which you can assign planes.

When you close this dialog, all instances belonging to the master of the selected instance are assigned to the selected plane.

Creating Planes and Assigning Them to Objects

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click on an instance in the schematic and select the Assign Planes option from the shortcut menu.

   The Assign Planes dialog appears, as shown in the following figure:



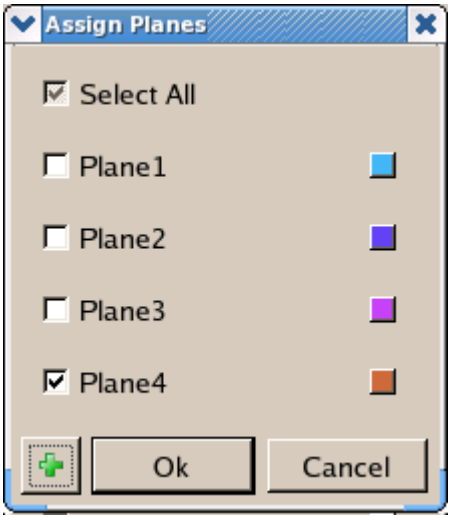
3. Click

4. in the above dialog.

   The Add New Plane dialog appears, as shown in the following figure:

5. Specify the plane name (say Plane4) in the above dialog.
6. Click the OK button.

   Plane4 appears in the Assign Planes dialog, as shown in the following figure:



7. Click the OK button in the above dialog.

After performing the above steps, GenSys assigns Plane4 to the selected object in the schematic.

Using Regular Expressions to Assign Planes

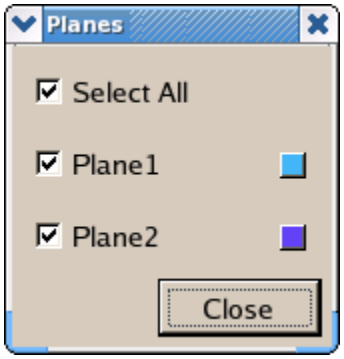
Use regular expressions in the add_to_plane Tcl command to assign a plane to multiple objects in one go.

For example, to assign the Plane1 plane to all the instances whose names start with the string i, run the following Tcl command:

```
add_to_plane -plane_name 'Plane1' -instance "i*"
```

Changing the Plane Color

Consider the following figure:



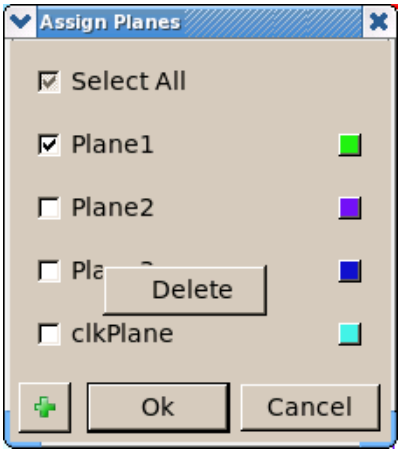
In the above figure, to change the color of the Plane1 plane, click the colored button adjacent to this plane. The Select color dialog from which you can choose a different color for that plane.

Deleting a Plane

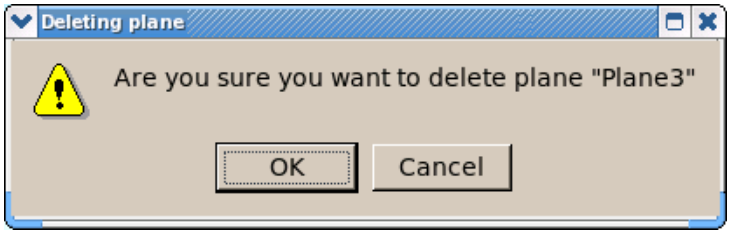
To delete plane, perform the following steps:

1. Enter into the edit mode. See Entering into the Edit Mode.
2. Right-click on an object, and select the Assign Planes option from the shortcut menu.

   The Assign Planes dialog appears.
3. In this dialog, right-click on the plane to be deleted and select the Delete option. See the following figure:



4. The Deleting plane dialog appears, as shown in the following figure:



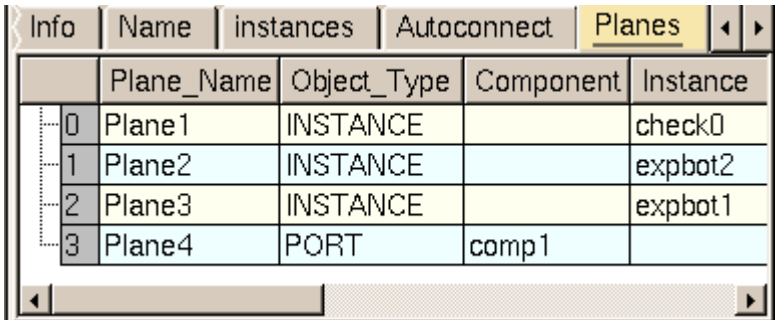
5. Click the OK button in the above dialog.

After performing the above steps, the selected plane is deleted.

Note:
Deleting a plane does not delete the objects that were assigned to that plane.

Using the Tabular View to Create Planes

Add planes in the Planes table of the currently loaded design, as shown in the following figure:



Note:
You can also add planes by using the add_to_plane Tcl command.

Validating Objects in the Planes Table

While adding or updating a row in the Planes table, if you specify an object that is not present in the design or specify an invalid object type for the specified object, GenSys highlights the corresponding cell in red color.

To validate rows containing such cells, right-click on the required row and select the Validate option from the shortcut menu. When you select this option, GenSys checks whether the specified object name and object type is valid. If not, GenSys highlights that row in red color.

Updating the Planes Table Based on Grouping/Ungrouping Operation

GenSys automatically updates the Planes table during grouping/ ungrouping.

For example, before ungrouping, if the name of an object in the Planes table is inst1.inst2 and after grouping, that object is grouped in the g1 group, GenSys automatically updates the name of that object in the Planes table to g1.inst1.inst2.

Later, if you ungroup that object, GenSys automatically updates the name of that object back to inst1.inst2 in the Planes table.

Note:
GenSys does not update rows containing cells with invalid information. For example, GenSys does not update rows containing wrong object information, such as object not present.

Notes

- At a time, you can view a maximum of 15 planes.

  However, it is possible to create any number of planes.

- GenSys does not display connections for which both end points lie in planes that are not selected.

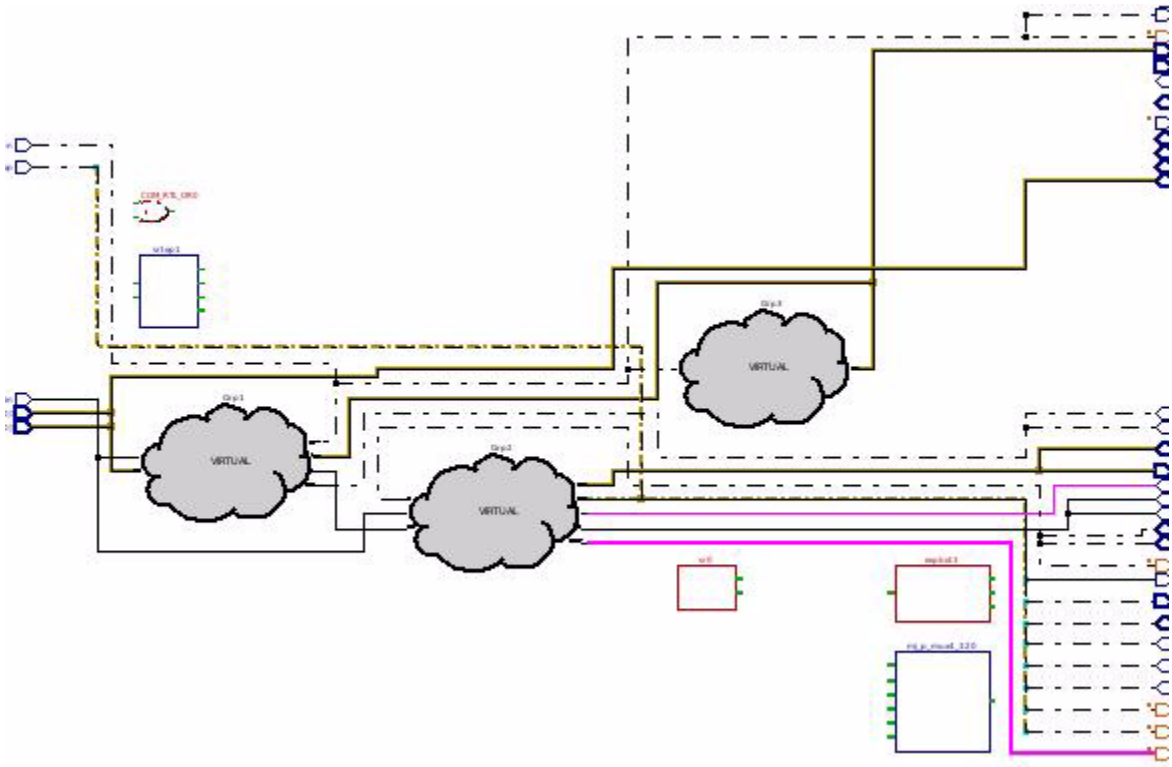
  For example, consider a connection between the I1.p1 and I2.p2 ports where I1.p1 is in the plane1 plane and I2.p2 is in the plane2 plane. In addition, I1 and I2 are not a part of any plane.

  Now, if both plane1 and plane2 are deselected, the connection between I1.p1 and I2.p2 will not be visible. However, I1 and I2 might be visible in this case as any other pin of both I1 and I2 might be a part of a selected plane.

- Changes made in the Planes panel are saved and applied in future sessions also.

Working with Virtual Groups

A virtual group is a logical collection of instances in a design. Each group appears as a cloud in the schematic, as shown in the following figure:



Virtual groups are essentially used during debugging of large designs. You can focus on specific areas of the design by hiding the other portions in a virtual group.

For details on working with virtual groups, refer to the following topics:

- Creating Virtual Groups
- Moving Instances in a Virtual Group
- Expanding a Virtual Group
- Dissolving a Virtual Group
- Renaming a Virtual Group
- Changing the Color of the Virtual Group
- Specifying a Default Color and Fill Pattern for a Virtual Group
- Redrawing a Virtual Group

Creating Virtual Groups

For small designs, you can select instances directly from the design and add them to a virtual group. For details, see Creating Virtual Groups in Small Designs.

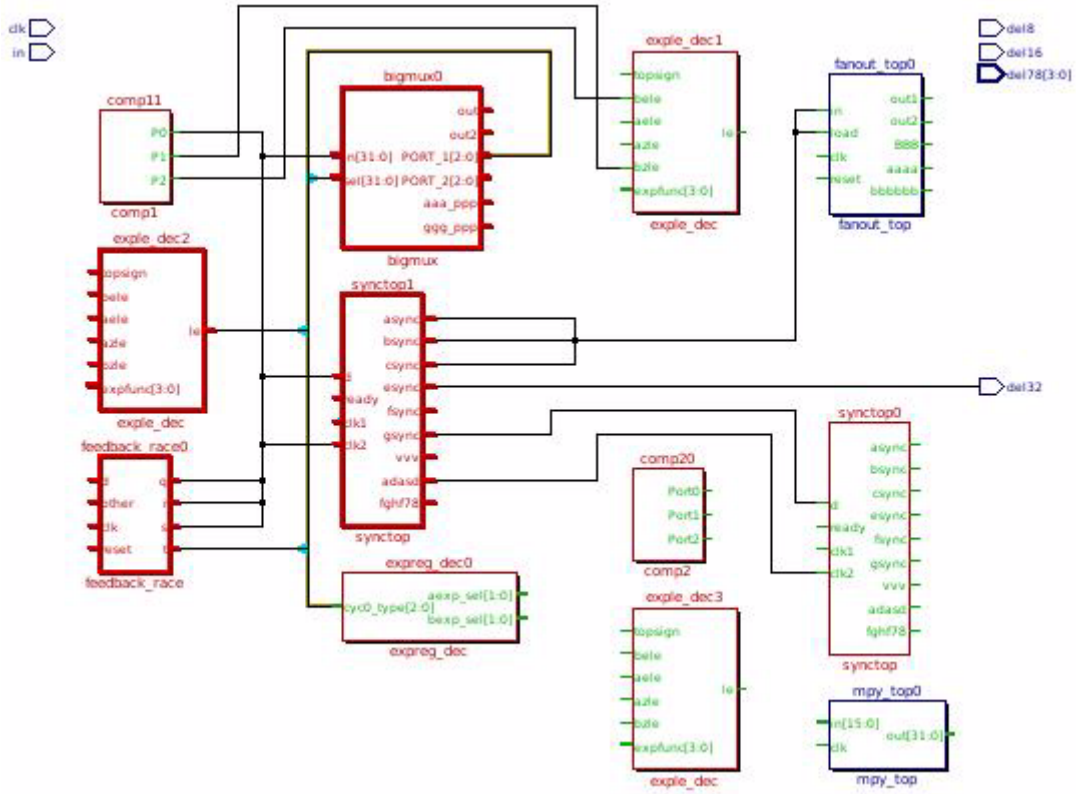
However, for big designs, it may be difficult to find the required instances to be added in a virtual group. For such cases, use the *Select Instances* dialog to add instances in a virtual group. For details, see Creating Virtual Groups in Big Designs.

Creating Virtual Groups in Small Designs

To create a virtual group in a small design, perform the following steps:

1. Click the

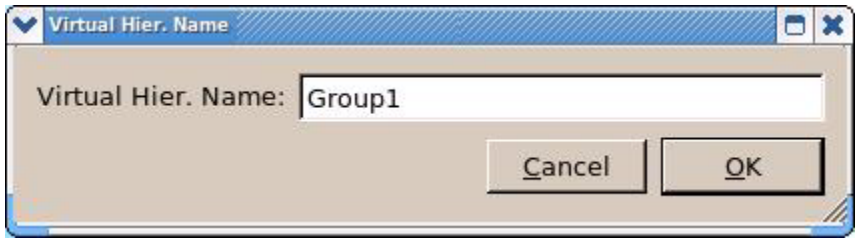
2. button in the canvas toolbar to enter into edit mode. For details, see Entering into the Edit Mode.
3. Click on the instances to be added in a virtual group by keeping the <*Shift*> keyboard key pressed.



4. Right-click in the schematic, and select the *Create New Virtual Hier* option from the shortcut menu.

   The *Virtual Hier. Name* dialog appears.

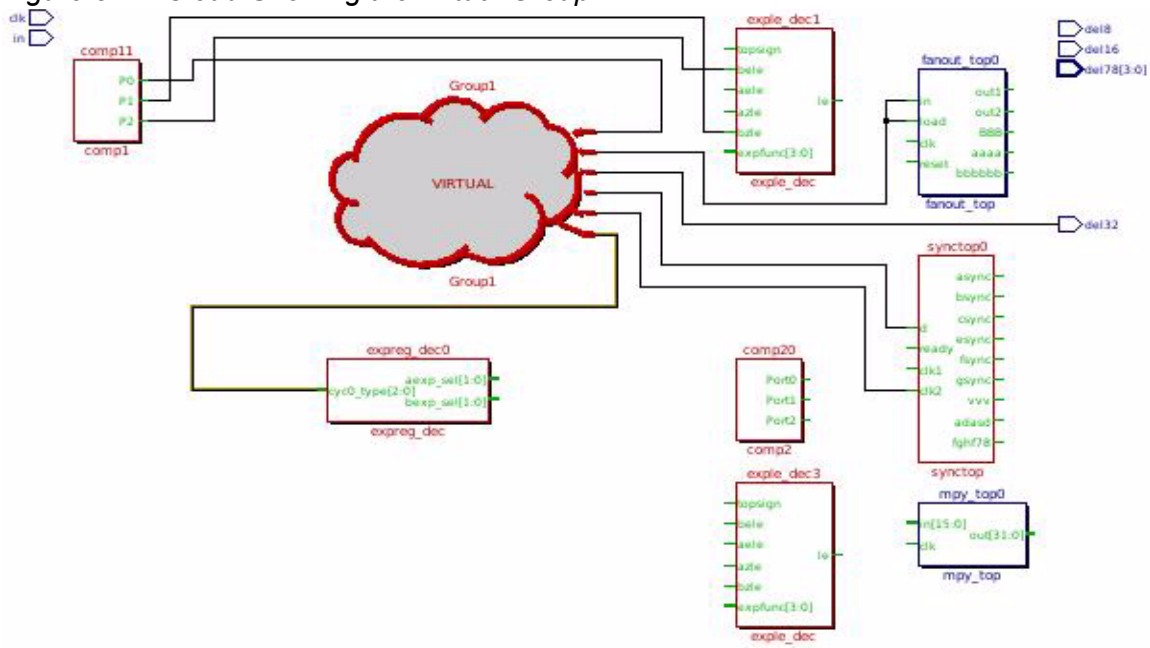
5. Specify the name of the virtual group in the *Virtual Hier. Name* dialog.



6. Click the *OK* button.

After performing the above steps, the selected instances are clubbed in the specified virtual group, and a cloud appears in the schematic showing the virtual group. See the following figure.

Figure 8-2 Cloud Showing the Virtual Group



Creating Virtual Groups in Big Designs

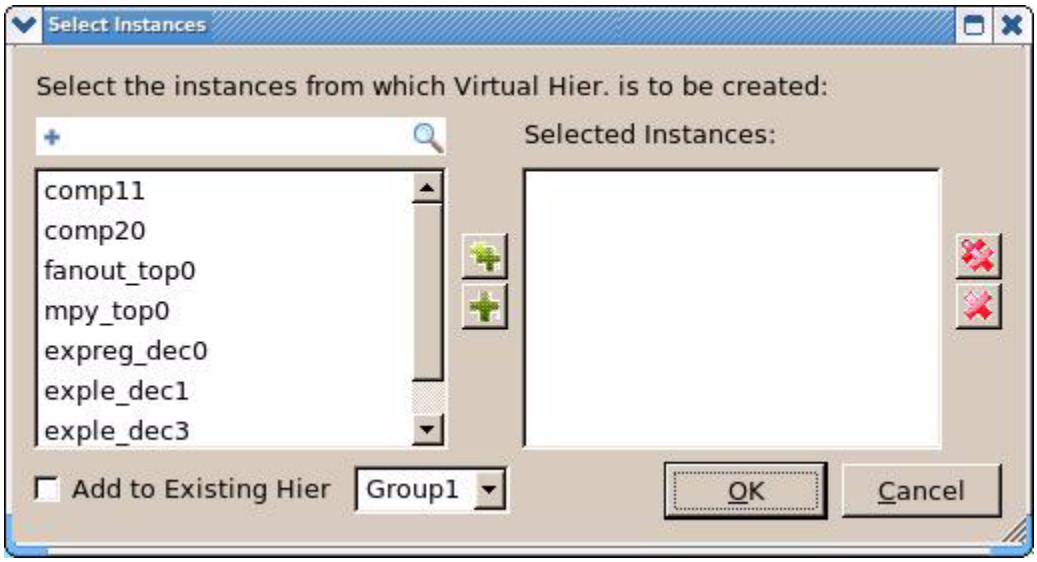
To create a virtual group in a big design, perform the following steps:

1. Click the

2. button in the canvas toolbar to enter into edit mode. For details, see Entering into the Edit Mode.

3. Right-click in the schematic and select the *Create New Virtual Hier* option from he shortcut menu.

   The *Select Instances* dialog appears displaying the names all the instances in the design. See the following figure:



4. In the above dialog, select the instances and click

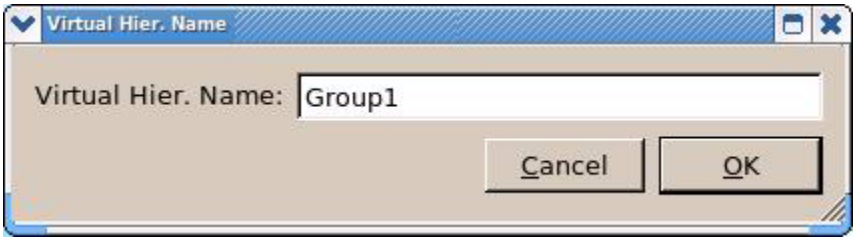
5. to add them in a virtual group.

   To add all the instances in a virtual group, click

   .

6. If you want to add the selected instances in an existing virtual group, select the *Add to Existing Hier* check box, select the name of the existing virtual group from the drop-down list, and click the *OK* button.
7. If you want to create a separate virtual group, click the *OK* button.

   The *Virtual Hier. Name* dialog appears.
8. Specify the name of the virtual group in the *Virtual Hier. Name* dialog.



9. Click the *OK* button.

After performing the above steps, the selected instances are clubbed in the specified virtual group, and a cloud appears in the schematic showing the virtual group. See figure Figure 8-2.

Moving Instances in a Virtual Group

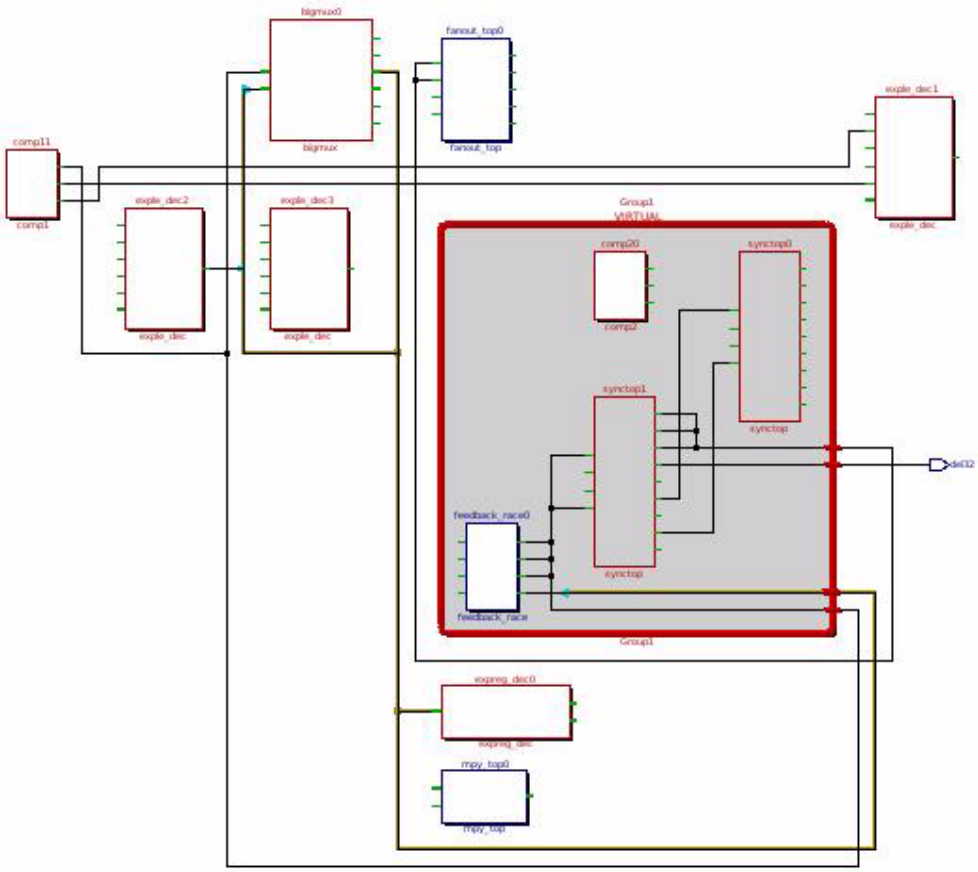
To move an instance in a virtual group, perform the following steps:

1. Select the instance to be moved in a virtual group.
2. Click <*Ctrl*>+*X* keyboard keys.
3. Select the virtual group in which the instance should be added.
4. Click <*Ctrl*>+*V* keyboard keys.

Expanding a Virtual Group

To view the instances of a virtual group, right-click on that group and select *Expand* from the shortcut menu.

The virtual group expands to show its instances, as shown in the following figure:



To collapse the expanded virtual group, right-click on it and select the *Collapse* option from the shortcut menu.

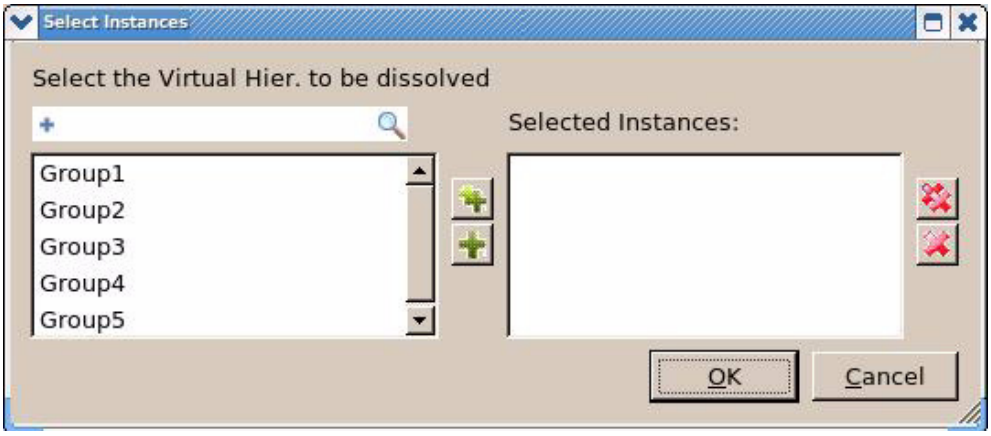
Dissolving a Virtual Group

In small designs, it is easy to locate the virtual groups to be dissolved. In such cases, right-click on the group appearing in the schematic and select the *Dissolve* option from the shortcut menu.

In big designs, the schematic is complex and it is difficult to find the virtual group to be dissolved. In such cases, perform the following steps:

1. Right-click in the schematic and select the *Dissolve Virtual Hier* option from the shortcut menu.

   The *Select Instances* dialog appears showing the list of virtual groups in the design. See the following figure:



2. In the above dialog, select the groups and click

3. .

   To dissolve all the virtual groups, click

   .

4. Click the *OK* button to dissolve the selected virtual groups.

Renaming a Virtual Group

To rename a virtual group, perform the following steps:

1. Right-click on the group and select the *Rename* option from the shortcut menu.

   The *Virtual Hier. Name* dialog appears.

2. Specify the new name in the *Virtual Hier. Name* dialog.
3. Click the *OK* button.

Removing an Instance from a Virtual Group

To remove an instance from a virtual group, perform the following steps:

1. Expand the virtual group. For details, see Expanding a Virtual Group.

2. In the expanded virtual group, right-click on the instance to be removed from that group and select the *Remove from Virtual Hier* option from the shortcut menu.

Changing the Color of the Virtual Group

To change the color of a virtual group, perform the following steps:

1. Right-click on the group and select the *Change Color* option from the shortcut menu.

   The *Change Color* dialog appears.

2. Select a color from this dialog.
3. Click the *OK* button.

Specifying a Default Color and Fill Pattern for a Virtual Group

Use the *Fill Color/Pattern* tab of the *GenSys:Options* dialog to specify a default color and fill pattern for a virtual group.

For details, see Updating the Color and Fill Pattern.

Redrawing a Virtual Group

Right-click on the virtual group, click the *Redraw* option from the shortcut menu, and select any of the following options from the submenu:

- *Complete Virtual Hier*: Select this option to properly place the instances in a virtual group to ensure best routing.
- *Nets*: Select this option to change the routing of instances, without moving the instances of the virtual group, to get an optimized routing.

Updating Color Settings and Miscellaneous Settings

To update color settings and specify miscellaneous settings, click the

icon in the canvas toolbar. Alternatively, right-click on the canvas view and select the Options option from the shortcut menu.

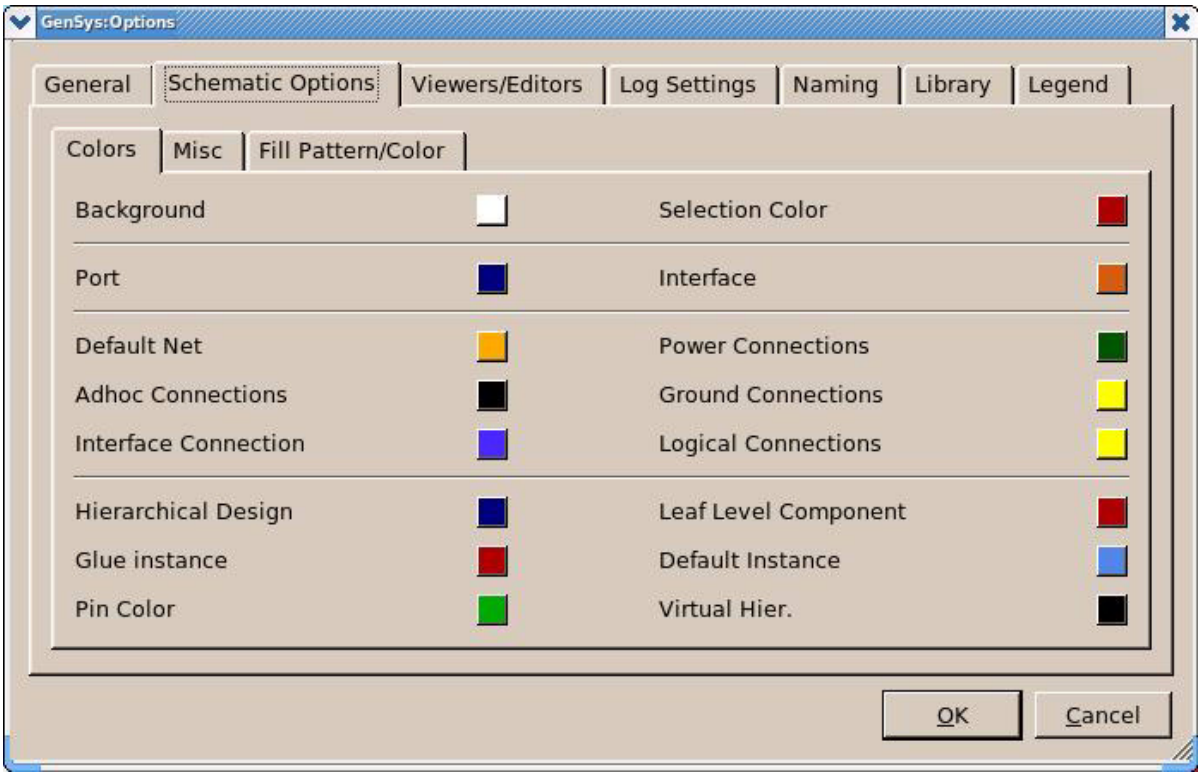
The Options dialog appears in which you can specify the required settings.

Updating Color Settings

You can change color for background, object types (ports, pins, and interfaces), and various types of connections.

To change color settings, click the Colors tab of the Schematic: Options dialog.

The following figure shows the page under the Colors tab:



In the above dialog, change the color settings as per your requirement, and click the OK button to apply the changes in the schematic.

Updating Color of an Object

To change the color of an object, perform the following steps:

1. Right-click on the object and select the Change Color option from the shortcut menu.

   The Change Color dialog appears.

2. Select a color from this dialog.

3. Click the OK button.

After performing the above steps, the selected object appears in the specified color in the schematic.

Restoring the Default Colors of Objects

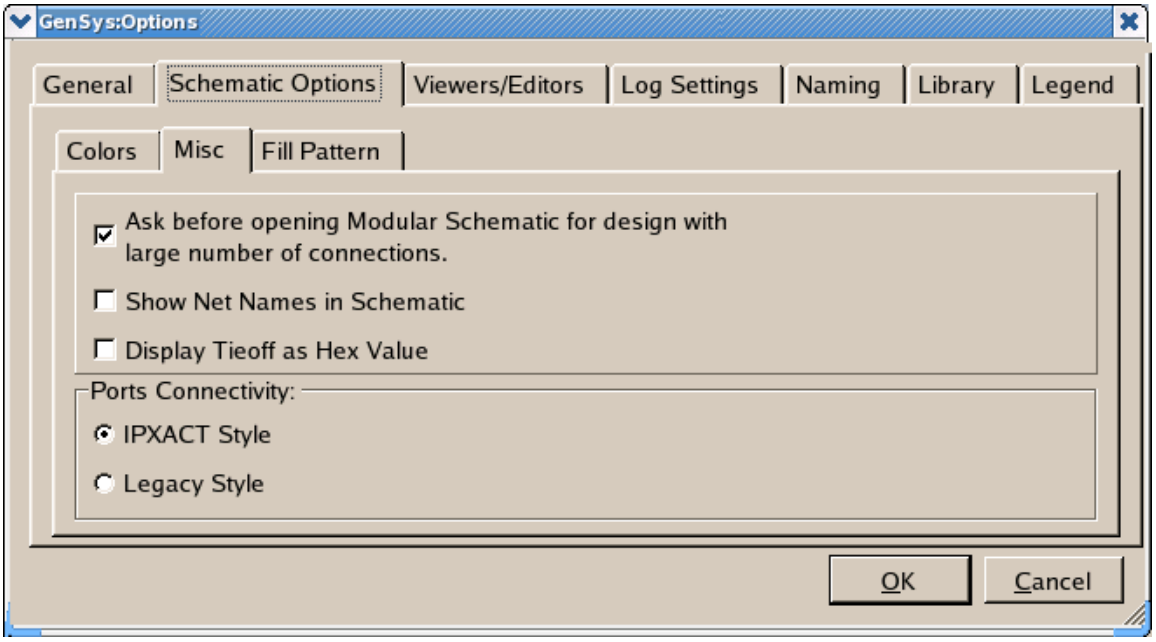
To restore back the default colors of all objects in the schematic, right-click in the schematic and select the Restore Default Colors option from the shortcut menu.

Updating Miscellaneous Settings

You can specify certain miscellaneous settings, such as showing/hiding MSB/LSB in schematic and specifying interface port connectivity style that should appear in the schematic.

To specify such settings, click the Misc tab of the Schematic: Options dialog.

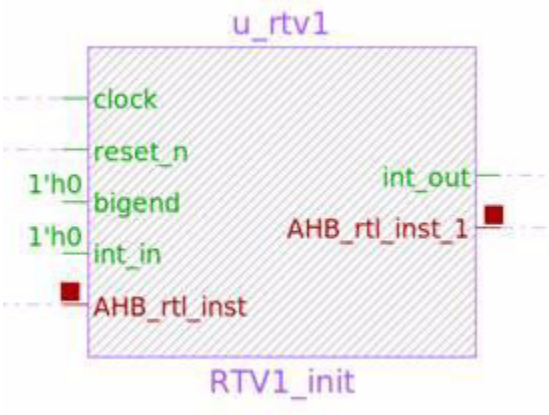
The following figure shows the page under the Misc tab:



In the above dialog:

- Select the Ask before opening Modular Schematic for design with large number of connections check box, if required.
- Select the Show Net Names in Schematic option to display net names in the schematic.
- Select the Display Tieoff as Hex Value option to show the hexadecimal value for a tieoff value.

  This is shown in the following figure:



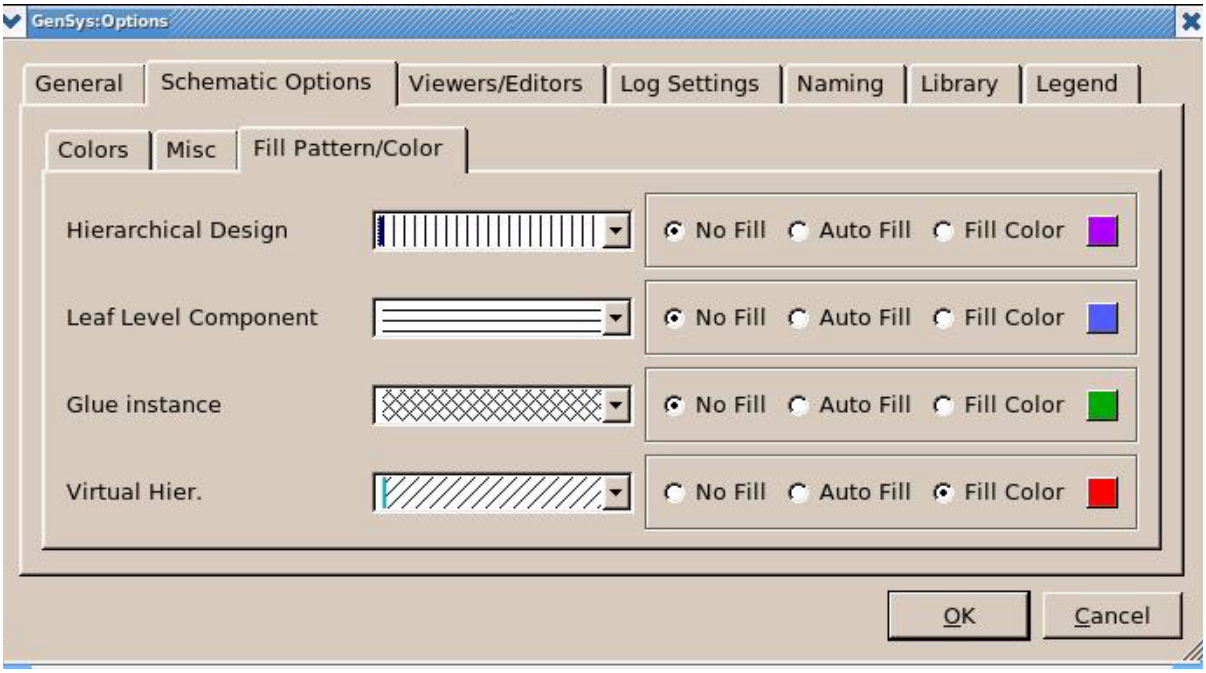
- Specify the interface port connectivity style that should appear in the schematic. Select any of the following options to specify a style:
  - IP-XACT Style: Under this style, master and mirrored slave appears on the right side and slave and mirrored master appears on the left side in the schematic.
  - Legacy Style: Under this style, master and mirrored master appears on the right side and slave and mirrored slave appears on the left side in the schematic.

Updating the Color and Fill Pattern

You can select a color and fill pattern of different types of objects, such as a hierarchical design, leaf-level components, glue instances, and virtual groups.

To specify a color and fill pattern of such objects, click the Fill Pattern/Color tab of the Schematic: Options dialog.

The following figure shows the page under the Fill Pattern/Color tab:

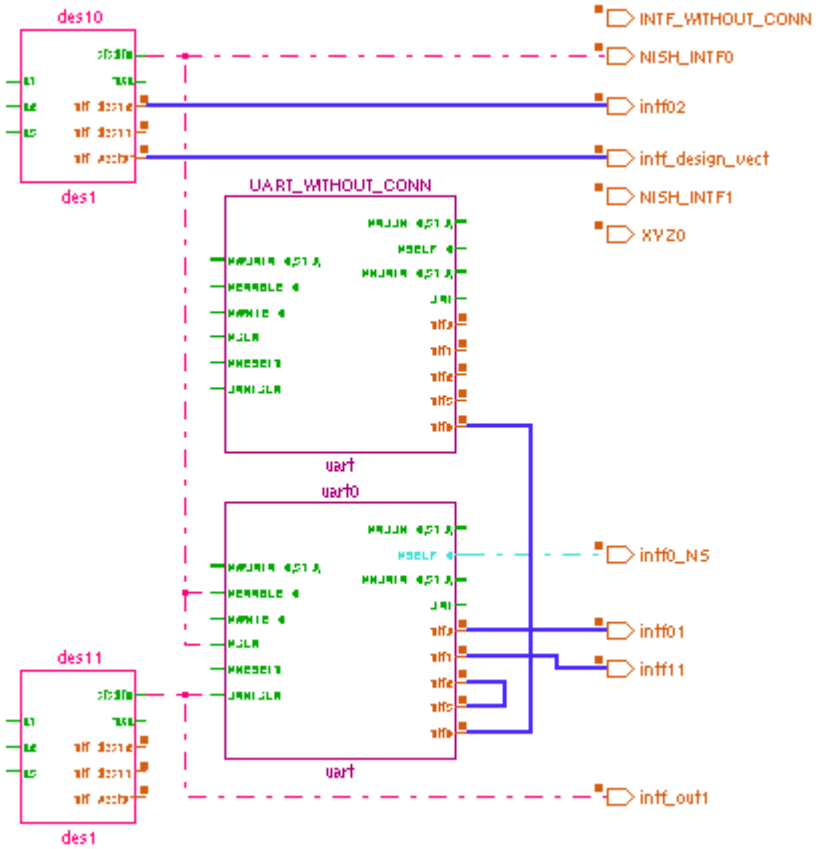


Expanding or Collapsing Interface Details

You can show interfaces and their connections in the schematic by clicking the

button in the canvas toolbar.

If there is an adhoc connection between a port of an interface and a design port, GenSys displays a partial connection between such ports in the form of a dotted line. This is shown in the following figure:



To view complete connections in such cases, click the

button.

When you click this button, it changes to

. You can click this changed button revert back to the previous view.

Changing Magnification in Schematic

You can modify the current magnification in the schematic by using the following buttons in the canvas toolbar:

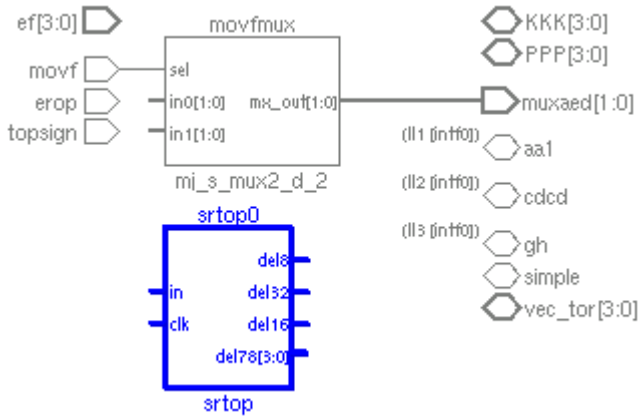
Button	Description
	Increases the magnification.
	Decreases the magnification.
	Fits the complete design view within the available schematic area.

Creating Focus on the Selected Object

It may happen that you want to work on a particular object through the schematic. In such cases, you can bring focus only on such objects by graying out other portions of the schematic, except that object.

To do so, select the required object and click the

button in the canvas toolbar. The following figure shows such a scenario in which the selected object is highlighted and rest of the design is grayed out:

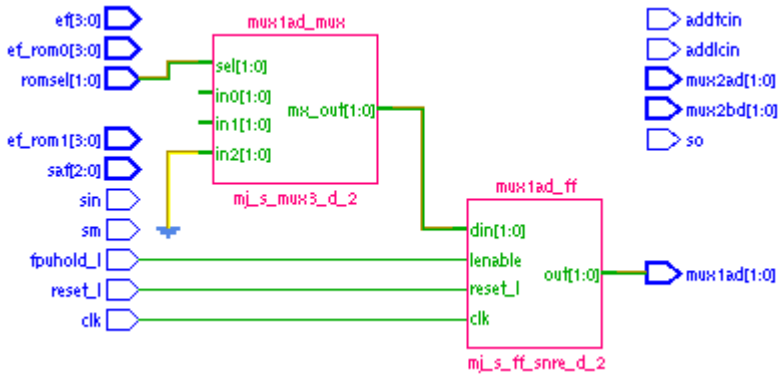


To go back to the previous view, that is remove the gray mode, select the

button in the canvas toolbar.

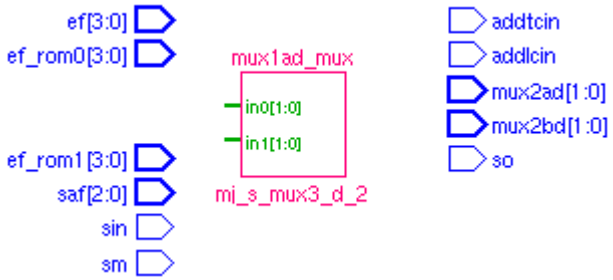
Viewing Only Unconnected Ports and Pins

Consider the following schematic view of a currently loaded design:



In the above schematic, to view only unconnected ports and pins, right-click in the schematic and select the Show unconnected Pins/Ports option from the shortcut menu.

When you select this option, the schematic view changes to the following:



To restore back to the previous view, right-click in the schematic and select the Show All Pins/Ports Connections option from the shortcut menu.

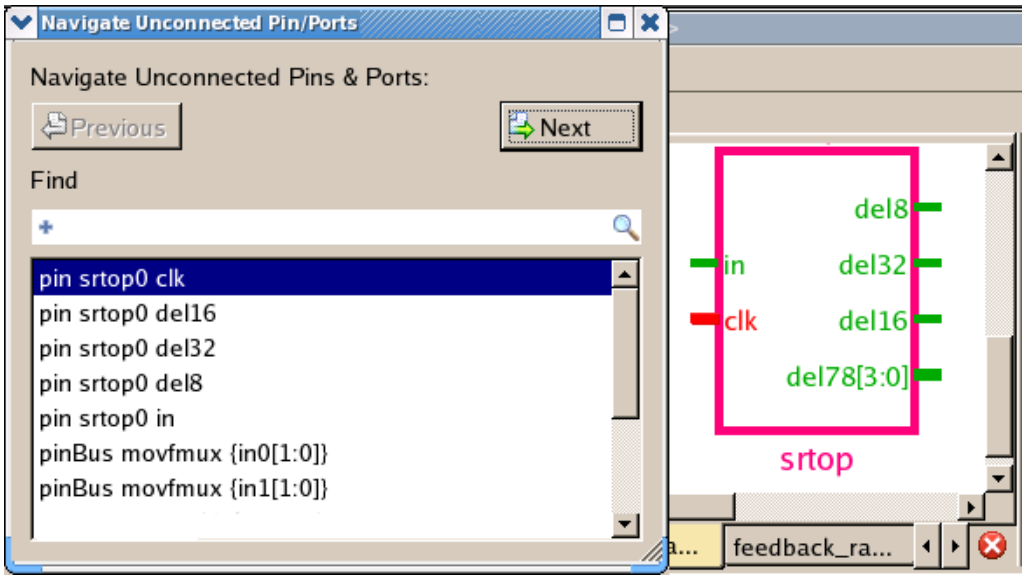
Navigating between Unconnected Ports and Pins

To navigate between unconnected ports/pins in the canvas view, click the

button in the canvas toolbar.

When you click this button:

- Portions of the design except for all unconnected ports and pins are grayed out.
- The Navigate Unconnected Pins/Ports dialog appears, as shown in the following figure:



By default, only one unconnected port/pin is focussed in the schematic.

To move to the next or previous unconnected port/pin, click the Next or Previous button, respectively, in the Navigate Unconnected Pins/Ports dialog.

To restore the previous view in which there is no highlighting of unconnected ports and pins, click

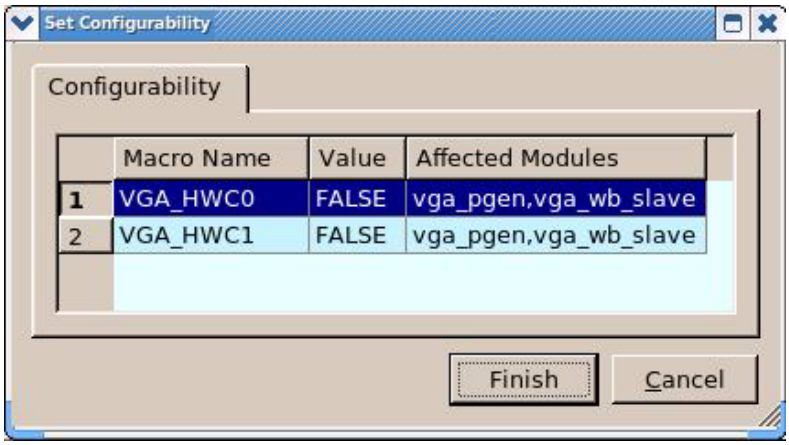
from schematic toolbar.

Modifying Configurability

To modify the configurability of instances, perform the following steps:

1. Enter in the Edit mode. For details, see Entering into the Edit Mode.
2. Select the instances by keeping the <*Shift*> keyboard key pressed.
3. Right-click on the selected instances and select the Set Configurability option from the shortcut menu.

   The Set Configurability dialog appears, as shown in the following figure:

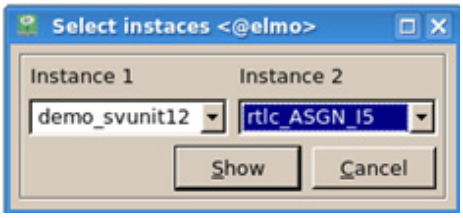


4. In the above dialog, modify the value of the macros as per your requirement.
5. Click the Finish button.

Viewing Connectivity between Two Instances

To view the connectivity between two instances, click

in the toolbar button. The following dialog appears:



In the above dialog, specify the names of the instances between which you want to view the connectivity.

Performing Undo and Redo

You can undo and redo changes in a particular GenSys session.

To undo a change, click the undo button (

) in the toolbar. Once you undo, the redo button (

) gets enabled to redo that change.

You can undo and redo changes any number of times until you exhaust all the available undo and redo actions.

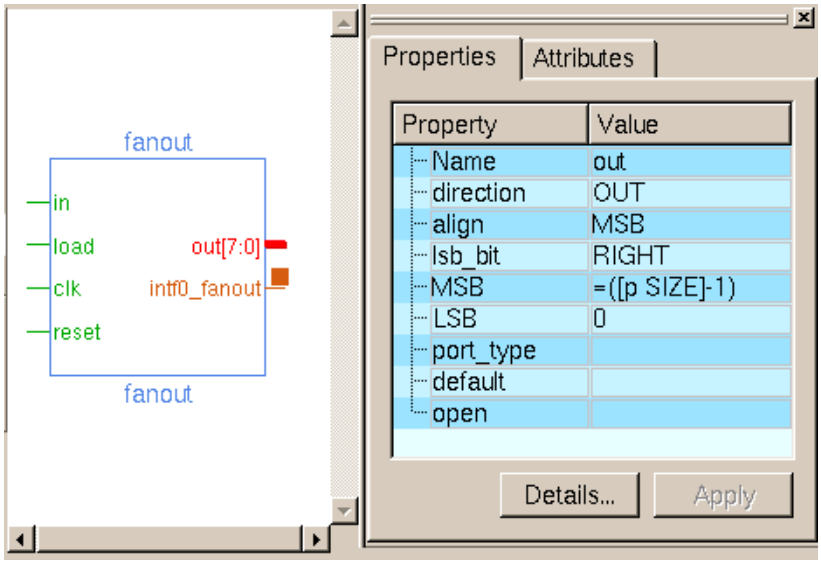
Using the Property Editor Pane

The Property Editor pane shows basic properties (in a tabular format) of the following types of objects:

Design	Component	Interface	Design Instance
Component Instance	Port		

You can modify these properties without the need to go specific GenSys tables in the tabular view.

When you select an object in the schematic, properties of that object appear in the Property Editor pane, as shown in the following figure:



Each row in the Property Editor pane represents one basic property.

Modifying Properties of an Object

To modify properties of an object, perform the following steps:

1. Click the

2. button in the canvas toolbar to enter into edit mode.
3. Click the required tab (Properties, Parameter, Attributes, or Configurability) in the Property Editor pane.
4. Modify a property.

   The modified properties appear in blue color.
5. Click the Apply button to apply the modified property on the selected object.

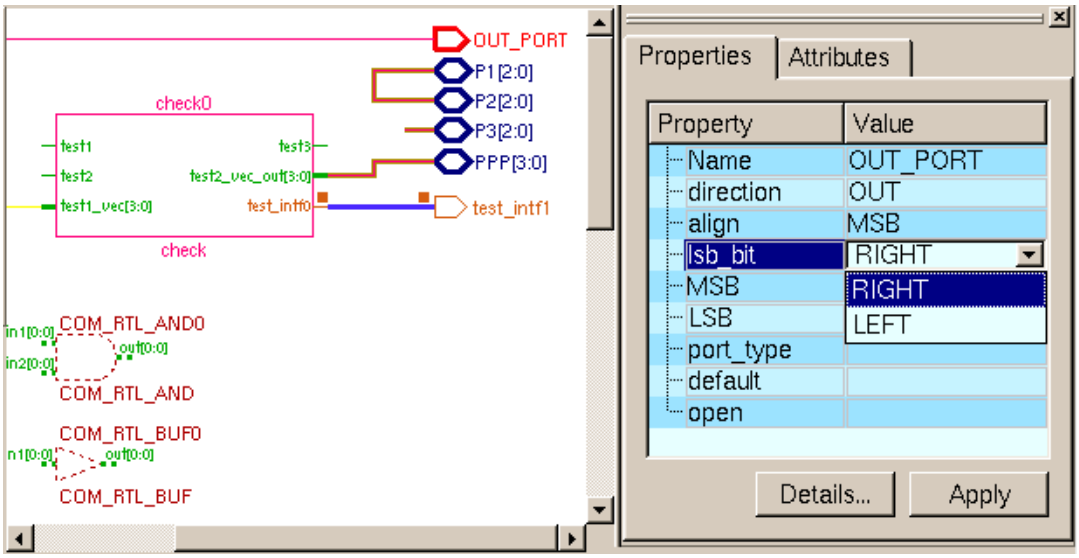
Modifying Different Types of Properties

You can modify different types of properties under various tabs, such as Properties, Parameter, Attributes, or Configurability.

Using the Properties Tab

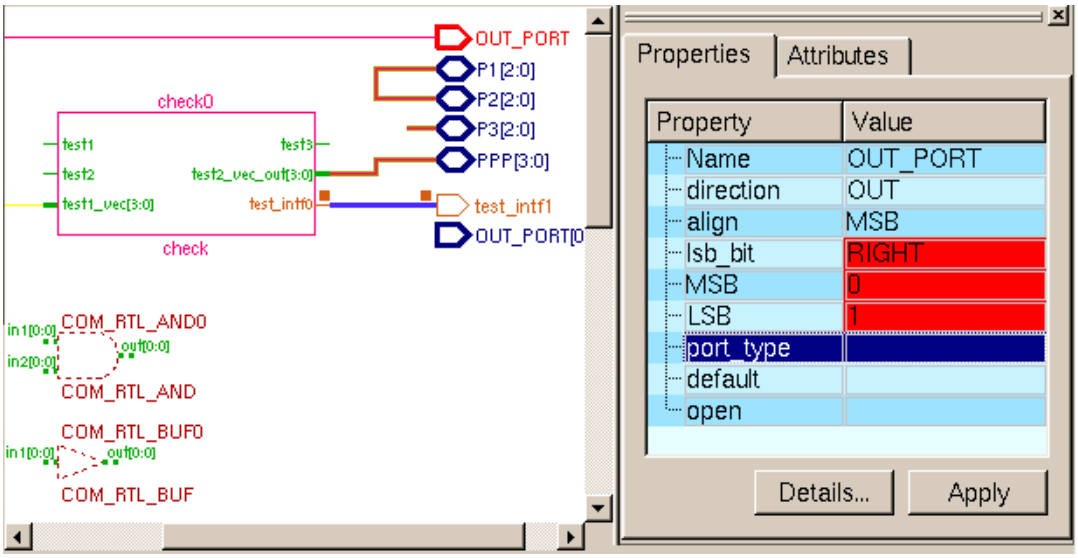
Under this tab, you can add or modify values of various properties of an object. Based on the type of selected object, different types of properties appear in the table under this tab.

For example, the following figure shows the properties of the selected port, OUT_PORT:

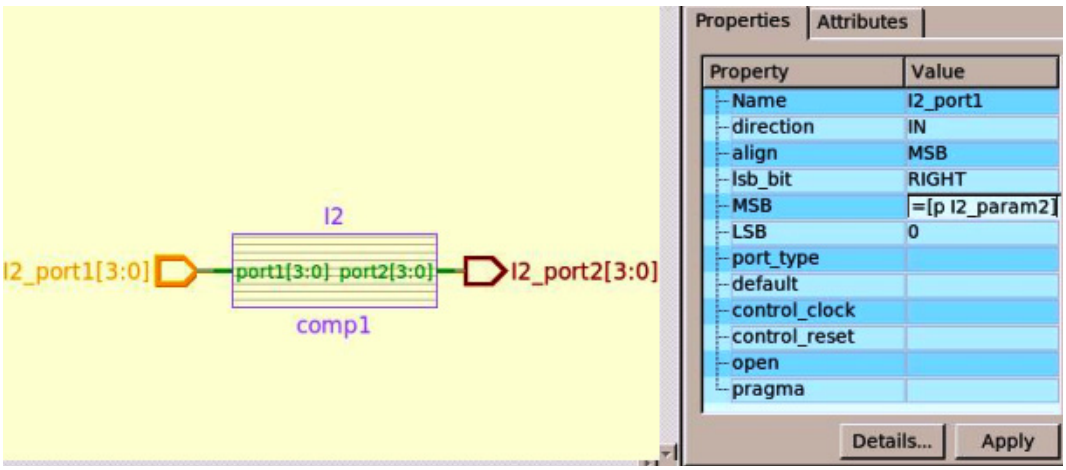


After updating the above table, click the Apply button to apply changes on the selected object.

When you click the Apply button, GenSys performs certain validation checks in this table. If any cell in this table contains an invalid value, the corresponding cells appear in the red color, as shown in the following figure:



You can specify expressions as the value of a property in the Properties tab. For example, in the following figure, the value of MSB of a port is specified in the form of an expression:



In the above figure, when you click the Apply button, GenSys evaluates the value of the expression and replaces that expression with its evaluated value.

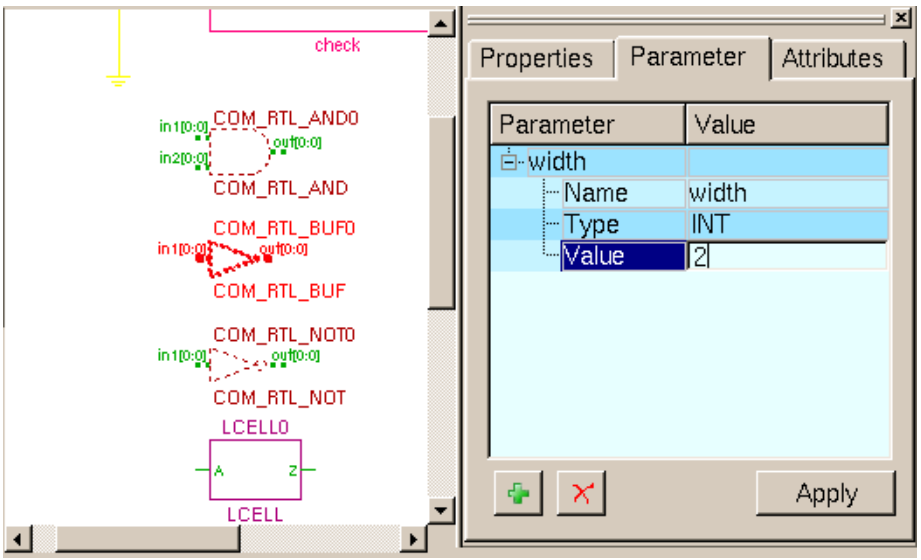
Note:

The table shown under the Properties tab is schema driven. All the fields shown under this tab are picked from the fields for which the SHOW_IN_PROPERTY_EDITOR personality is set in the schema view.

Using the Parameters Tab

Under this tab, you can add or modify parameter details, such as parameter type and value. However, you cannot change the parameter name as it is read-only.

For example, consider the following figure showing table under the Parameters tab for the selected instance, COM_RTL_BUF0:

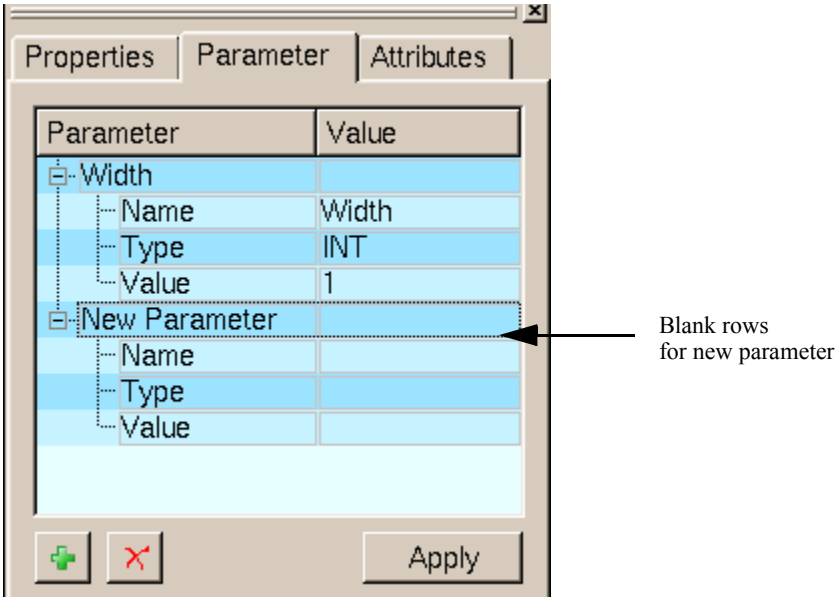


To add a new parameter, perform the following steps:

1. Click the

2. button.

   Blank rows appear under the Parameters tab, as shown in the following figure:



3. Specify new parameter details, such as name, type, and value.

   If you want to use an expressions to specify a parameter value, enclose the expression within curly braces, as shown in the following example:

   ```
   {= [p WIDTH] - 1}
   ```

4. Click the Apply button.

After performing the above steps, the new parameter is added for the selected object.

To delete a parameter, select the parameter node, and click

button.

Note:

The Parameter tab appears for a component, component instance, design, and design instance. In case of a component instance and a design instance, you can edit only parameter values.

Using the Attributes Tab

Under this tab, you can add or modify attribute details, such as attribute name, its type, and value.

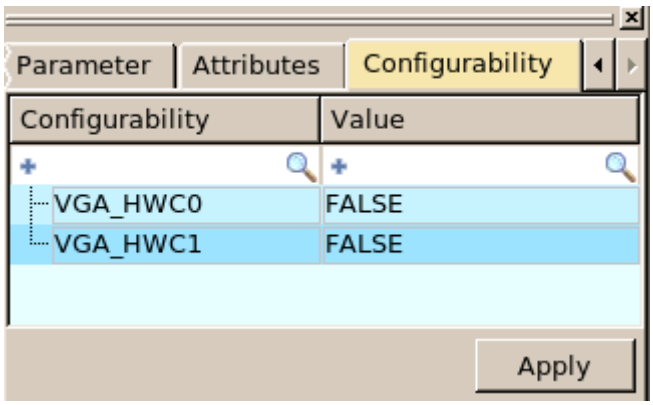
Adding or modifying attributes is similar to adding or modifying parameters. For details, see Using the Parameters Tab.

Based on an object selected, attributes are applied on that object.

Using the Configurability Tab

Under this tab, you can configure the macros of the selected instance.

When you select an instance in the canvas view, all its macros appear under the Configurability tab, as shown in the following figure:



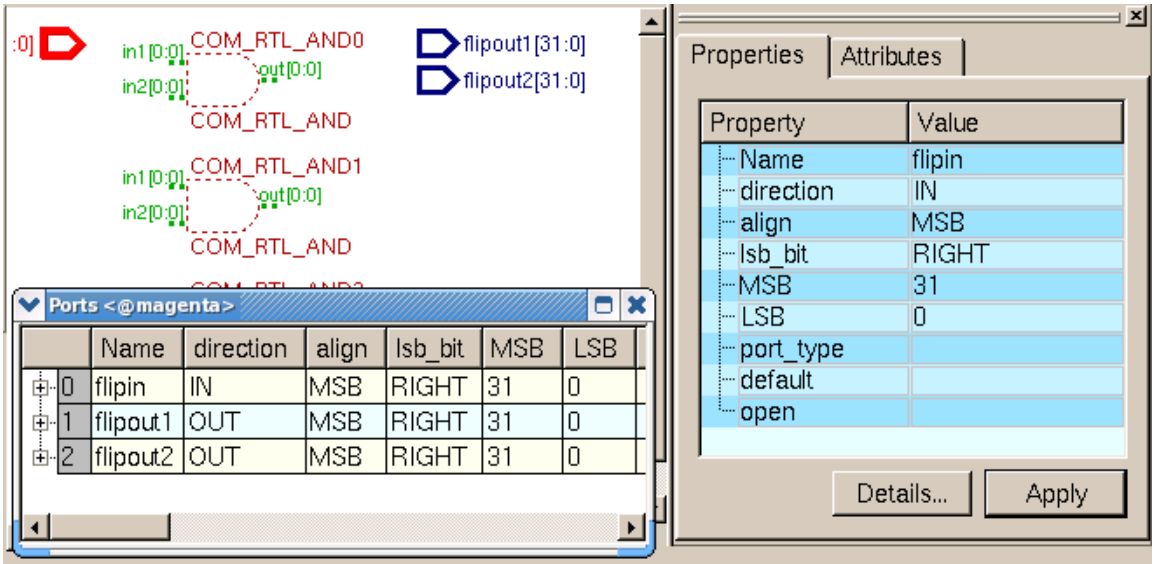
In the above table, set the macros to TRUE or FALSE, and click the Apply button.

To configure macros at a global level, select the Configurability/Visibility option of the Tools Menu.

Viewing Property Details of an Object

Based on the type of object selected, when you click the Details button, an appropriate table (from the tabular) view appears in a separate window showing the details of that object.

For example, when you select a port in the schematic and then click the Details button in the Property Editor pane, the Ports table appear. This is shown in the following table:



Tables Displayed Based on the Object Type Loaded or Selected

Based on the type of object selected or loaded, appropriate tables are displayed when you click the Details button. This is mentioned in the following table:

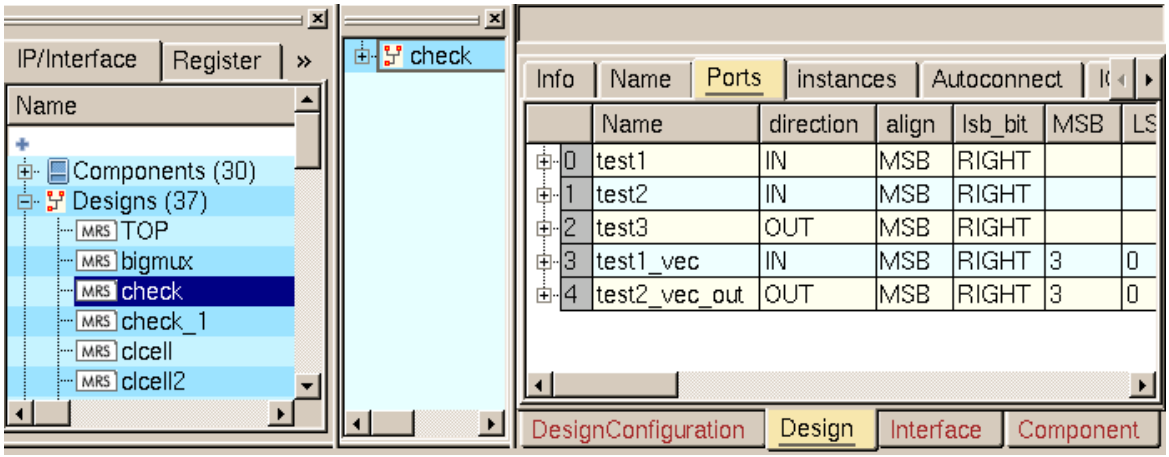
Object Type	Action	Table Displayed
Design	Loaded	Open info table for object
Component	Loaded/ Selected	Open info table for object
Interface	Loaded	Info table for object
Interface Class	Selected	If the corresponding component is loaded, the Interface Matching widget appears. Else, the Details button is disabled.
	Loaded	Info table for object
Component Instance	Selected	Details button disabled
	Loaded	Info table for object

Object Type	Action	Table Displayed
Interface Instance	Selected	If top level design shown in modular schematic then launch interface matching widget else disable
Ports	Selected	Ports table
Terminals	Selected	Details button disabled
Logical Ports	Selected	logical ports table

Using the Tabular View

In the tabular view, all information of the currently loaded object is present in different tables.

For example, the following figure shows the Ports table of the currently loaded design:



The above view consists of various tabs representing tables. You can modify data in these tables as per your requirement.

Understanding the Tabular View

When you click on any tab, the corresponding table appears below that tab. In addition, the tab color is highlighted in yellow to indicate the tab selection.

Each tab may have sub-tabs to categorize different types of information.

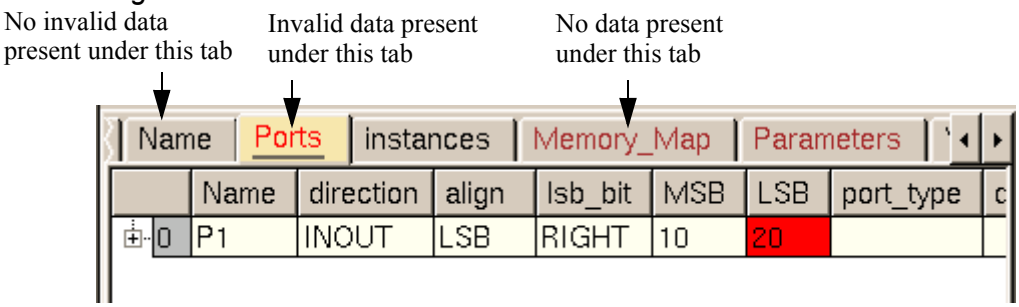
Understanding Tab Coloring

The text color of the tab can be any of the following:

- Black: This indicates that tables under that tab contains no invalid information.
- Brown: This indicates that there is no information under that tab.
- Red: This indicates that some invalid cell is present under any level of the table within that tab.

The following figure shows different tab coloring:

Figure 8-3 Tab Coloring

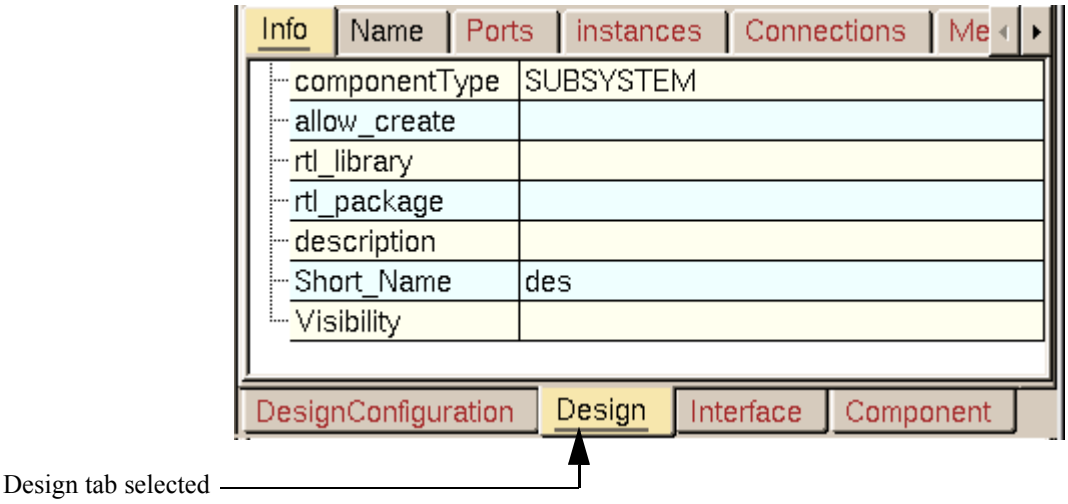


Understanding Tabs of a Particular Object Type

At the bottom of the tabular view, four tabs are present, namely DesignConfiguration, Design, Component, and Interface.

This is shown in the following figure:

Figure 8-4 Tabs of Different Object Types



Working in the Tabular View

You can use various options in the tabular view to perform different actions.

Inserting Rows

To insert rows in the Data Table window, perform ANY of the following actions:

- Right-click in the Data Table window, and select the Insert Row menu option.
- Click in the Data Table window, and press the Insert key.
- Click the Insert Row icon (

-) from the toolbar.

After inserting rows in the Data Table window, you can enter the required values in the cells for each row. To add/edit information in a cell, select the required cell (this displays the cell in the edit mode), and enter information in that cell.

Moving Through Cells

GenSys enables you to move across various cells of a table by using the following keys:

- TAB key to move to the adjacent cell on the right of a row

- SHIFT+TAB key to move to the adjacent cell on the left of a row
- Up arrow key to move to the above cell in the current column.
- Down arrow key to move to the below cell in the current column.

Tables that start with a plus sign icon expand into sub-tables. For example, you can add bitfield definitions under each row for a register. Press the Enter key to expand a sub-table.

Fields have automatic basic entry edit/check. For example, string and integer have standard entry fields, enumerated types have drop-down combo boxes, and descriptions have a WYSIWYG editor.

In description fields, where HTML widget is shown, if you CTRL+click on any word, a web browser is opened displaying the clicked word, which is supposedly a web location.

This web browser can be set from Tools->Options dialog.

Number fields can be selected to have either decimal or hex entries.

Deleting Rows

To delete a selected row in the Data Table window, perform ANY of the following actions:

- Right-click and select the Delete Row menu option from the shortcut menu.
- Select the Delete Row toolbar button (

-).

This displays the Confirm Delete dialog. Click the Yes button to delete the selected rows.

Copying/Pasting Data

Tables in GenSys contain editable cells, editable drop-down lists, and non-editable drop-down lists. You can copy/paste data in these cells/lists as described below.

In Editable Cells

You can copy/paste data from and into cells by using the CTRL+C and CTRL+V keys, respectively.

If you copy a single cell and then select multiple cells (of a single column) to paste the data, all the cells will display the new value.

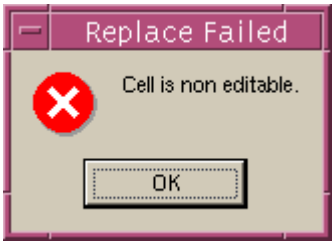
If you copy contents from multiple rows (say, four rows) and paste them to a single cell, GenSys will copy the contents into the next four rows (starting from the selected row).

In Non-editable Drop-down Lists

You can copy/paste content from and into non-editable drop-down lists by using the CTRL+C and CTRL+V keys, respectively. However, only those contents from the lists are copied that are present in the drop-down lists.

If you try to paste content into a drop-down list, and if that content is not present in the drop-down list values), you will get the following message:

Figure 8-5 Replace Failed Dialog



In Editable Drop-down Lists

You can copy/paste content from and into editable drop-down lists using the CTRL+C and CTRL+V keys.

When pasting data into editable drop-down lists, you need to first double-click the editable drop-down list cell and then paste the data.

Note:
Editing the cells is determined by the specifications given in the Schema.

In Read-only Cells

You cannot copy/paste any data from and into read-only cells.

Following tabs in the Design View are read-only:

- The Tieoff tab and Intentional Open tab in the Connections table.
- The Memory_Map_View tab in the HW_SW.Memory_Map table.

Following tabs in the Component View are read-only:

- The Register_Map tab in the Memory table.

Specifying Live Formula using Tcl Expressions

You can enter Tcl expressions to assign values to cells in tables. To enter a Tcl expression, click the formula icon (

) in any selected cell. You can use arithmetic operator or mathematical function such as log, log2, to perform the calculations. You can also perform calculations based on the value in a column of a subtable or a parent table.

Live formula expressions would be evaluated using Tcl interpreter and would support pure Tcl language syntax.

Note:

For backward compatibility, GenSys also supports specification of live formula expressions using Perl expressions. To learn more, refer to Specifying Live Formula using Perl Expressions.

The methods to enter Tcl expressions to specify live formulae are described below.

Using Simple Formulas

You can assign values to cells by performing simple calculations using arithmetic operators or functions, such as +, -, *, /, log, sqrt, etc., in any selected cell. For example, =17*1024.

Note that it is required to start each Tcl expression with an = sign. Expressions given without using = sign would not be evaluated in Tcl based expression flow and would be assumed to be simple string values.

Using Parameters and Attributes

You can use the Tcl routine, p, to use value of a parameter.

For example, the following Tcl expression will return the value of parameter param1.

```
=[p param1]
```

You can also use dependent parameters. For example, you can create a parameter pb whose value is dependent on the value of parameter pa, as shown below:

```
add_parameter -name pa -type INT -value 4
add_parameter -name pb -type INT -value {=[p pa]*4}
```

To use value of an attribute, you can use the Tcl routine, a.

For example, the following Tcl expression will return the value of attribute attr1.

```
=[a attr1]
```

The Tcl routine, a, searches the specified attribute in the current COM object associated with the cell. If not found, attribute with the given name would be searched in its parent COM object, upto the top level object, component, design, or interfacedef.

Note:

If an expression is syntactically incorrect or the given parameter or attribute does not exists, actual expression is displayed in the cell and is highlighted in RED color.

Accessing Value of a Table cell

You can use the Tcl routine, celldata, to access value of a table cell. The syntax to access values of cells in the same row is:

```
[celldata <cell-name>]
```

For example, if you want the value of MSB of a port to be three times the value of LSB, click the formula icon and enter the following expression in the MSB cell of the port:

```
=[celldata LSB]*3
```

Consider another example. If you want to specify that the value in LSB should be square root of the value of MSB, then click the formula icon and enter the following expression in the selected cell:

```
=[sqrt [celldata MSB]]
```

Specifying Absolute Path

The syntax to specify the absolute path of a cell is:

```
<top>.<tab>.<tab>.<column>
```

where,

<top> is design, component or interface

<tab> is a table or subtable name (or hierarchical list of tab names)

<column> is a name of the column whose value is accessed

For example, the following Tcl expression will return the value of MSB of port p1.

```
=[celldata component.Ports(Name=p1).MSB)]
```

In the example given above, the expression returns the value of MSB column where the value in the Name column of the Ports table is p1. You can also specify this expressions the follows:

```
=[celldata component.Ports(p1).MSB)]
```

In this case, the value p1 will be taken from the key column specified in schema.

To specify conditional expression for row selection you must use parenthesis,(), as shown in the example given below.

```
=[celldata design.instance(i1).attributes(attr1).Value]
```

Accessing Subtable Data

You can also access data present in a subtable of a particular row by using Tcl expressions.

For example, if you want to access the msb value of that row of the model sub-table for which the language column has the value a, and you want to assign that msb value in the

MSB column of the Ports table, specify the following Tcl expression in the MSB column of the Ports table:

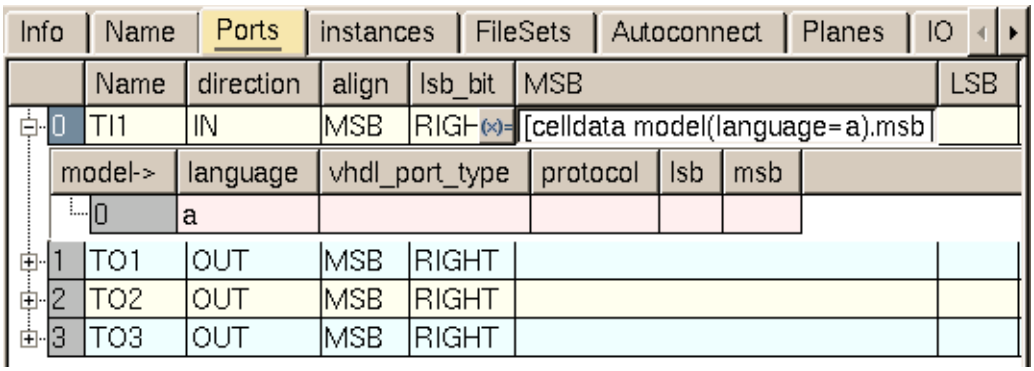
```
=[celldata model(language=a).msb]
```

In the above expression, model is the name of the subtable. language is the name of the column of the row where the required msb is present and whose value we need to access.

msb is the name of the column in which the required value is present that needs to be accessed.

Refer to the figure shown below:

Figure 8-6 Tcl Expression for Accessing Subtable Data



If a column is a key column, you can specify the Tcl expression in the following way also:

```
=[celldata
<sub-table>(<column-key-value>).<column-name-whose-value-need-to-be-acces
sed>]
```

Note:

The table names and column names specified in the Tcl expression are case-sensitive.

Accessing Parent Table Data

You can also access the data present in a parent table. For example, if you need to access the value of MSB of bitfield b1 in the RegisterBlock subtable of the Memory tab, you need to give the following:

```
=[celldata ../Memory.RegisterBlock(R2).bitfields(b1).MSB]
```

where, ../ indicates that we need to move to the parent table, Memory is the name of the table where RegisterBlock subtable is present, bitfields is the subtable present in the RegisterBlock table.

Specifying Relative Path

You can specify the path of the cells that need to be accessed, relative to the current cell. In the examples shown in the above sections, the relative path specification had been used. Consider another example.

If the position of the current row is the MSB field in the Ports table, and you need to access the MSB value in the bitfields subtable present inside the RegisterBlock subtable in the Memory tab, specify the following:

```
=[celldata ../Memory.RegisterBlock(R2).bitfields(b1).MSB]
```

where, based on the position of the current row, you access the parent table (../), then access the Memory tab (Memory), then the register R2 in the RegisterBlock subtable, then access the bitfield b1 in the bitfields subtable, and finally access the MSB value of bitfield b1.

Using Tcl Variables

You can use Tcl variable in a Live-formula, but it is important to identify the scope in which the Tcl expression live-formula is required to be executed.

If it is required to execute the Tcl expression immediately, instead of using live-formulae it is recommended to use Tcl expression commands, as shown in the following example:

```
set x 4add_port -name p_in1 -direction IN -msb [expr 2*$x] -lsb 0
```

If you want to use delayed execution of Tcl expression (when the generator gets the value of the cell in which live formula is stored), it is recommended not to use local Tcl variables as they might not be available when Tcl-expression is being evaluated. In such cases, it is better to use variables with complete scope, using the :: Tcl operator, as shown in the following example:

```
set ::var1 10add_port -name p_in2 -direction IN -msb {=$::var1 * 2} -lsb
0
```

where, $::var1 refers to variable var1 defined in global scope.

Global variables can be defined in .gensys.setup file and then can be used in a live-formula as described above.

Limitations in using Tcl Live Formula Expressions

There is one limitation in using Tcl live formula expressions. Currently, complex parameters are not fully supported. This limitation will be resolved in future releases.

Specifying Live Formula using Perl Expressions

Note:

By default, you can specify live formula using Tcl expressions. However, if you want to use Perl expressions (old flow), specify the -use_perl_expr command-line option.

Note that when you are accessing the value of some other cell, you need to use the $ sign before the column name that is being accessed. For example, if you want the value of MSB of a port to be three times the value of LSB, click the formula icon and enter the following expression in the MSB cell of the port:

```
$LSB*3
```

Accessing Subtable Data

You can also access data present in a subtable of a particular row.

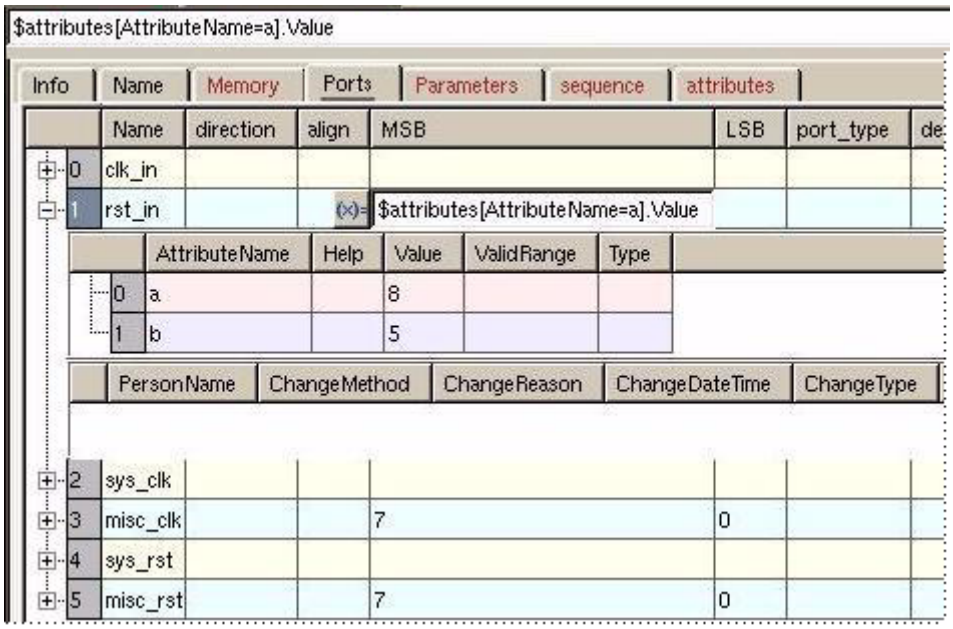
For example, if you need to access the value of attribute a of a port rst_in, and put that value in the MSB column of the port, you need to give the following Perl expression:

```
$attributes[AttributeName=a].Value
```

where attributes is the name of the subtable. AttributeName is the name of the column where the attribute is present and whose value we need to access. In this case, the attribute name is a.

Value is the name of the column where the value of the specified attribute is present. (You are accessing the value/data present in the Value column corresponding to attribute a). Refer to the figure shown below:

Figure 8-7 Perl Expression for Accessing Sub-Table Data



The Perl expression mentioned above can also be written as,

$attributes[a].Value

This is because, the AttributeName column is the key column in the attributes subtable. Therefore, we need not specify the column name of the attribute, if the column is a key column in the schema.

Note:
The tablename, column name specified in the Perl expression are case- sensitive.

Accessing Parent Table Data

You can also access the data present in a parent table. For example, if you need to access the value of MSB of bitfield b1 in the RegisterBlock subtable of the Memory tab, you need to give the following:

```
$../Memory.RegisterBlock[name=R2].bitfields[ShortName=b1].MSB
```

where ../ indicates that we need to move to the parent table, Memory is the name of the table where RegisterBlock subtable is present, bitfields is the subtable present in the RegisterBlock table.

Specifying Relative Path

You can specify the path of the cells that need to be accessed, relative to the current cell. In the examples shown in the above sections, the relative path specification had been used. Consider another example.

If the position of the current row is the MSB field in the Ports table, and you need to access the MSB value in the bitfields subtable present inside the RegisterBlock subtable in the Memory tab, specify the following:

```
$../Memory.RegisterBlock[name=R2].bitfields[ShortName=b1].MSB
```

where, based on the position of the current row, you access the parent table (../), then access the Memory tab (Memory), then the register R2 in the RegisterBlock subtable, then access the bitfield b1 in the bitfields subtable, and finally access the MSB value of bitfield b1.

Specifying Absolute Path

You can also specify the absolute path of the cells, as given below:

```
$component.attributes[c].Value
```

where component is the component at the top-level, attributes is the name of the column where the attribute is present. c is the name of the attribute and Value is the name of the column where the attribute value is present and needs to be accessed.

If a top-level design contains more than one component, you need to specify the component name in the Perl expression, as given below:

```
$design.component[comp2_inst_1].attributes[B].Value
```

Again, if the component contains more than one interface, you need to specify the interface name in the expression, as given below:

```
$design.component[comp2_inst_1].interface[intf1].logical_ports[ActualPort
Name=P1].MSB
```

Using Tabular View Right-Click Menu Options

The following menu options appear when you right-click any of the tables in the tabular view:

Cut	Copy	Paste	Expand Row
Collapse Row	Expand All	Collapse All	Freeze Row
UnFreeze Row	Insert Row	Insert Sub-Table	Delete Row
Evaluate	Validate	Dependency Graph	Configure
Find Text	Flat Table View	Group View	

Cut

This menu option is used to cut the currently selected rows in a table. The keyboard shortcut for this menu option is CTRL+X.

Copy

This menu option is used to copy the currently selected rows in a table. The keyboard shortcut for this menu option is CTRL+C.

GenSys allows you to copy a complete row with or without sub tables. If you select the row header and select the Edit -> Copy menu option, the entire row with its sub tables get copied. While pasting, GenSys prompts you to specify if you want to past the row with sub tables or without sub tables.

Paste

This menu option is used to paste the already cut or copied rows in other tables. The keyboard shortcut for this menu option is CTRL+V.

Expand Row

This menu option is used to expand the selected row to open its sub table. The option is enabled only for a row that has a sub table.

Collapse Row

This menu option is used to close the sub tables for the selected row.

Expand All

This menu option is used to open all the sub tables for the table item over which this feature was called.

Collapse All

This menu option is used to close all the sub tables for the table item over which this feature was called.

Freeze Row

This menu option is used to freeze the selected row so that this row remains in view and all the rows below can be scrolled. When a row is frozen, the position of all the rows above it are also frozen.

UnFreeze Row

This menu option is used to unfreeze the selected row so that it can be scrolled. This option is enabled only in the shortcut menu of a frozen row.

Insert Row

This menu option is used to insert a new row with default values after the current selected row in a table. The keyboard shortcut for this menu option is Insert key.

Insert Sub-Table

This menu option is used to insert a subtable in the selected row (for example, bitfields for a register, bitEnumValues for a given bitfield). The keyboard shortcut for this menu option is CTRL+T.

Delete Row

This menu option is used to delete all the selected rows from the table in focus.

Evaluate

This menu option is used to evaluate the entries of the individual dynamic tables.

Validate

This menu option validates the entries for the individual tables for which the validation function is provided in the schema.

Dependency Graph

This menu option is used to facilitate specifying the dependencies in such a way that you do not need to remember the syntax of dependency definition. As a lot of dependencies are the same for different registers or fields, ease of copying and pasting in the text editor would be very helpful in creating dependencies from already defined dependencies. While you are typing, context sensitive help is shown at appropriate places i.e. help is shown depending upon the text you are entering.

Configure

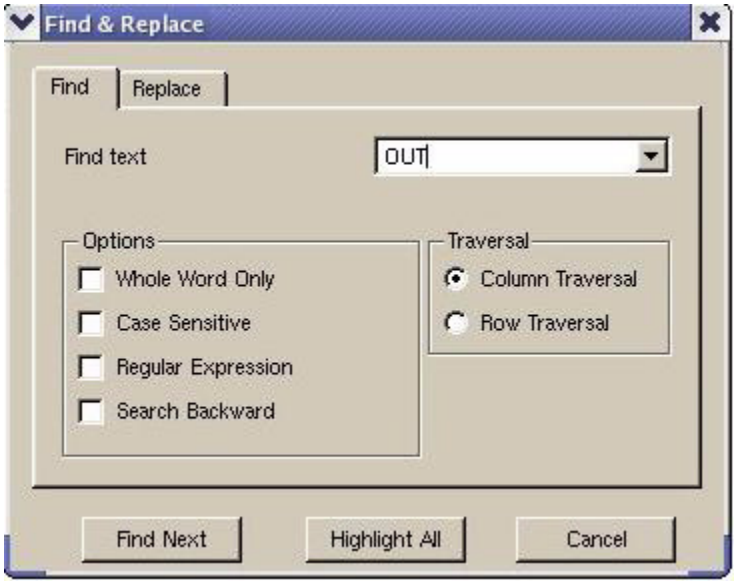
This menu option is used to hide or unhide columns, hide or unhide sub-tables, sort columns, and specify key columns. When you select this option, the Configure Data Table dialog appears, which is also shown when Edit > Configure... menu options is used.

Find Text

This menu option is used to search for items that matches the entered text.

When you select this menu option, the following dialog box appears. Enter the item that needs to be searched in the Find text field. Check mark the options that matches your requirement and click the Find button. The Highlight All button highlights all the occurrences of the item searched.

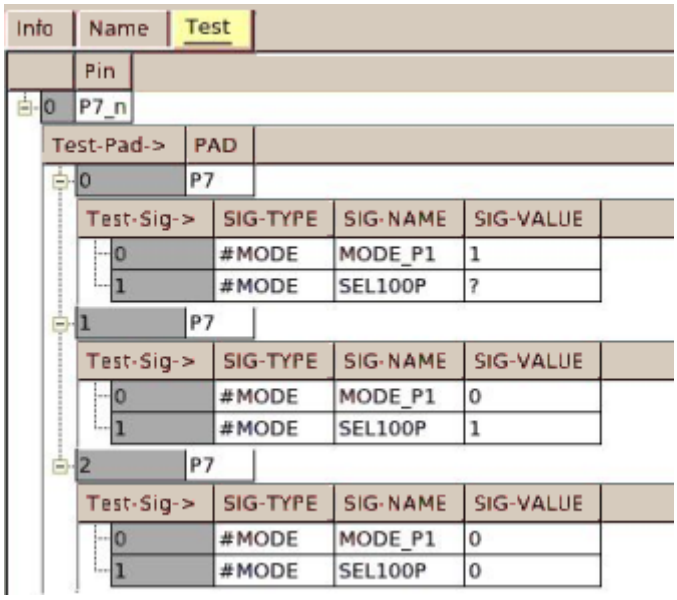
Figure 8-8 Find & Replace Dialog



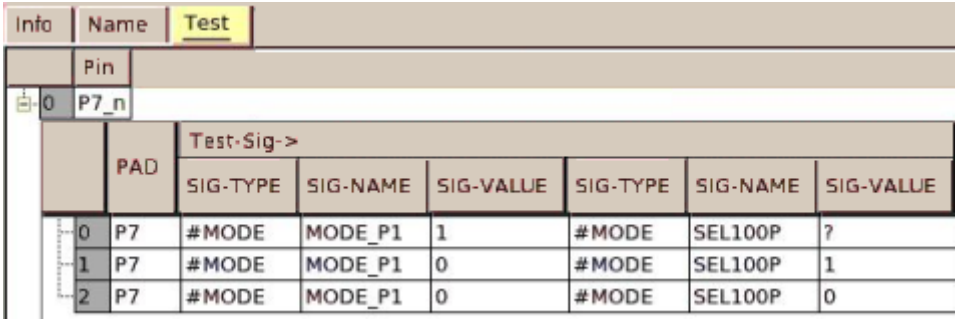
Flat Table View

This menu option is used to view the flat table view of a hierarchical table. In this view, sub tables appear as additional columns. You can further configure this view to hide or show selected columns.

For example, consider the Test table shown in the following figure:



When you right-click on the above table and select the Flat Table View option from the shortcut menu, this table gets flattened, as shown in the following figure:

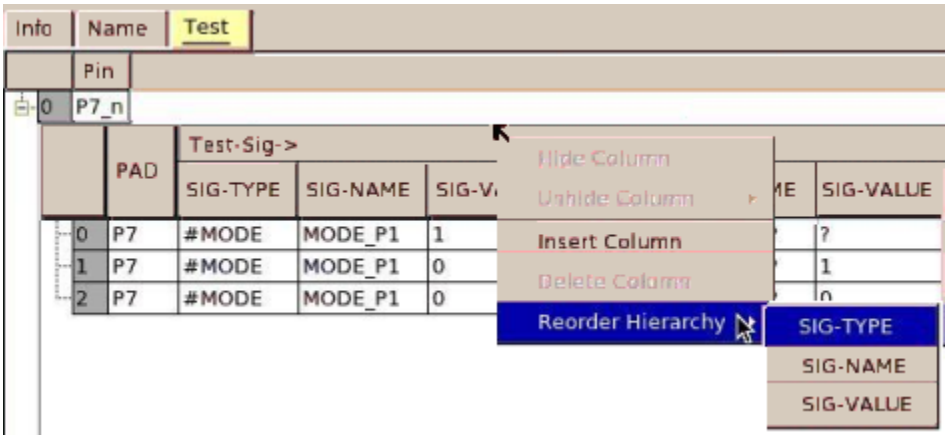


In the above flattened view, you can reorder the hierarchy of the table.

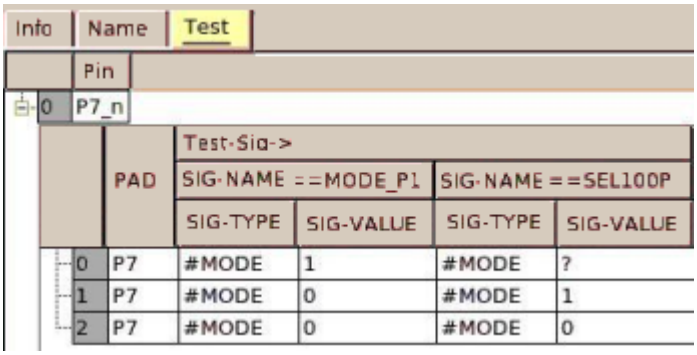
Note:

This is not a recommended methodology. Therefore, it will be phased out in the next two releases.

To reorder the hierarchy, right-click on the above table and select the Reorder Hierarchy option from the shortcut menu, as shown in the following figure:



From the above sub-menu of the Reorder Hierarchy option, select the option based on which you want to reorder the hierarchy. For example, when you select SIG-NAME from the above sub-menu, the table hierarchy changes to the following:

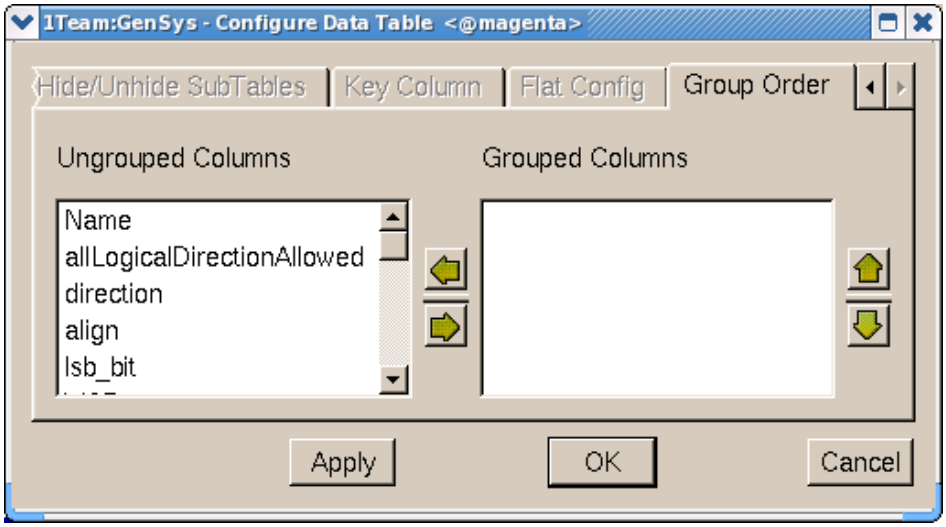


Group View

This menu option is used to view the data of a table grouped on the basis of data in selected column(s). This option is especially helpful for tables that have multiple levels of hierarchy.

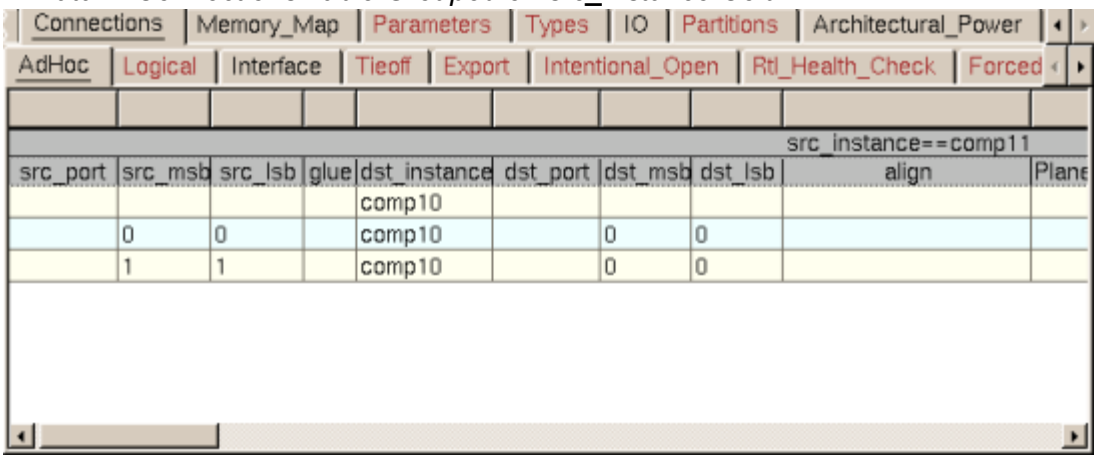
When you select this menu option, the table can be configured for group view settings. Next, right-click anywhere in the table and select the Configure menu option. The Configure Data Table dialog is displayed, as shown in the following figure.

Figure 8-9 Configure Data Table Dialog



This dialog lists the names of all the columns in the Ungrouped Columns list. Select the column on which you want to group the data and click OK. For each unique value in the selected column, the data in other columns in grouped. The figure given below shows data in the Connections table grouped on the src_instance column.

Figure 8-10 Data in Connections Table Grouped on src_instance Column



You can reconfigure the group view to group the data on multiple columns.

Using Table Column Menu Options

The following menu options appear when you right-click any of the columns of a table:

Hide	Unhide	Views	Set Precision
Sort	Insert Column	Delete Column	Filter Options
Remove Filter			

Hide

This menu option is used to hide the currently selected column.

Unhide

This menu option is used to unhide the currently hidden columns of the table. When you click this option, the list of currently hidden columns is displayed, from which you can select the column that you wish to unhide.

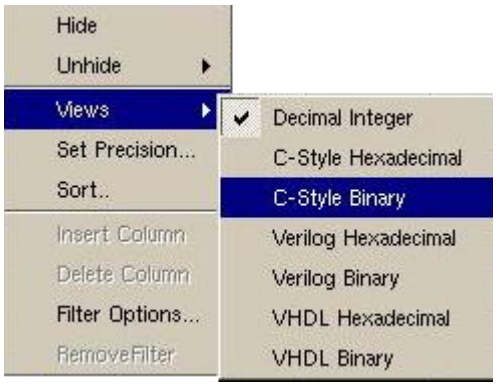
Views

Note:

This menu option is enabled for a column only when the type of that column, as specified in the schema (with the TYPE :: keyword), is either INT_TYPE or HEX_TYPE.

This menu option is used to select the view format of the column values. Clicking this menu option lists all the available views for the column values, as shown below:

Figure 8-11 Views Menu Option



You can select a view from the listed options to change the display format of the column values.

Set Precision

Note:

This menu option is enabled for a column when the type of that column, as specified in the schema (with the TYPE :: keyword), is either INT_TYPE or HEX_TYPE.

This menu option is used to set the precision (in bits) for the column values. Clicking this option displays the Set Precision dialog, as shown below:

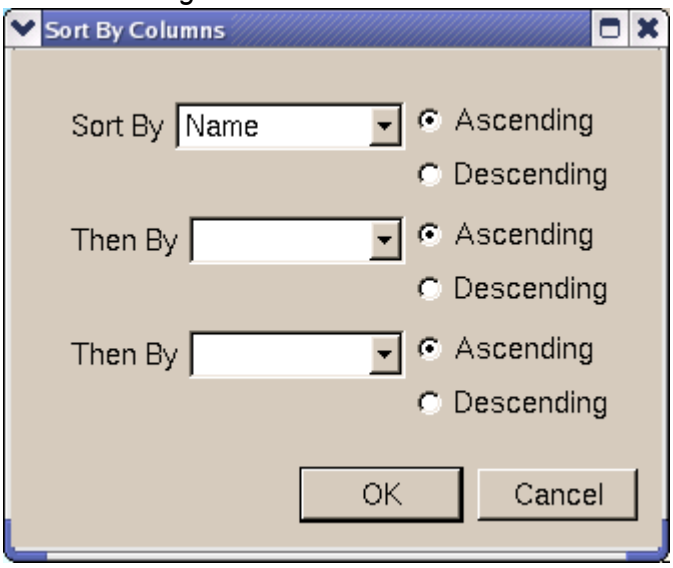
Figure 8-12 Set Precision Dialog

Here, you can specify the precision (in bits) for the column values.

Sort

This menu option is used to sort a table. When you select this menu option, the Sort By Columns dialog box appears, as shown below:

Figure 8-13 Sort By Columns Dialog



Here, you can specify maximum of three columns to sort the table. Sorting depends upon the order in which the columns are specified. By default, a column is sorted in ascending order. However, to sort a column in descending order, select the Descending option adjacent to that column.

In sorting, alphanumeric characters of strings, such as PSEL13 and PSEL[13] are considered and entries are sorted correctly.

Insert Column

Note:

This menu option is enabled only for the View Tables having hierarchical columns.

This menu option is used to insert a new child column under the selected parent column.

Delete Column

Note:

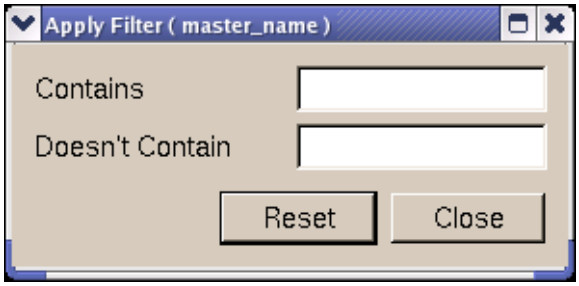
This menu option is enabled only for the View Tables having hierarchical columns.

This menu option is used to delete the selected child column from the parent column.

Filter Options

This menu option is used to apply a filter on the selected column. When you select this menu option, the Apply Filter dialog appears, as shown below:

Figure 8-14 Apply Filter Dialog



Here, you can specify one or both of the following filter options:

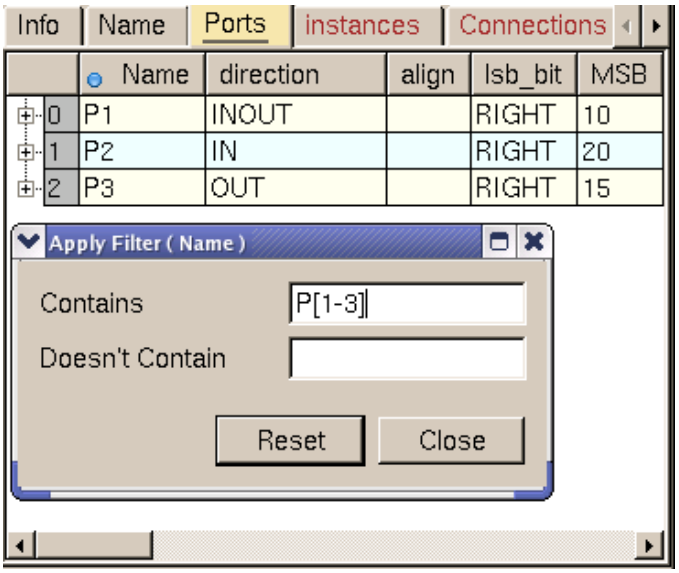
- Contains: To display the column entries that contain the string/Perl regular expression specified here.
- Doesn’t Contain: To display the column entries that DO NOT contain the string/Perl regular expression specified here.

As you start specifying the filter criteria in the Contains and/or Doesn’t Contain fields, GenSys automatically starts filtering the results accordingly in the selected column. When you close the Apply Filter dialog, the filtered results are retained, and the filter icon,

, appears on that column to indicate that a filter has been applied on that column.

You can specify the filter criteria as a plain text or a regular expression. For example, to display the entries starting with p1 to p3 in the Name column, you can specify the range as p[1-3] in the Contains field, as shown in the following figure:

Figure 8-15 Applying Filter



In the above figure, note that the Name column gets automatically filtered based on the filter criteria specified in the Apply Filter dialog.

You can reset the specified filter by clicking the Reset button.

Following are some more examples for specifying Perl regular expression to apply filter on a column:

- To filter those values that do not start with des, either specify the Perl regular expression, ^des, in the Doesn't Contain text box or specify ^(?!des) in the Contains text box.
- To filter those values that start with des, specify the Perl regular expression, ^(?=des), in the Contains text box.

Remove Filter

This menu option is used to remove the currently applied filter from the column.

Using Tab Menu Options

The following menu options appear when you right-click any tab:

Hide	Unhide	Evaluate	Validate	Refresh

Hide

This menu option is used to hide a tab.

Unhide

This menu option is used to display a list of tabs that are currently hidden. You can choose the tab from the list that needs to be unhidden.

Evaluate

This menu option is used to evaluate the table and display the evaluated table.

Validate

This menu option is used to validate all the entries in the table.

Refresh

This menu option is used to refresh a table.

Using Keyboard Shortcuts

This section provides a list of shortcut keys that can be used in GenSys GUI:

Press	To...
CTRL+N	Create a design, interface, or a component
CTRL+O	Open an existing design, interface, or a component
CTRL+S	Save the changes made to the currently active design, interface, or component
CTRL+X	Cut the contents of the selected cells
CTRL+C	Copy the contents of the selected cells
CTRL+V	Paste the contents of (copied/cut) cells to the selected cells
CTRL+SHIFT +C	Clear the contents of the selected cell(s)
CTRL+A	Select all the cells in the table
ALT+A	Perform AutoFill

Press	To...
INS	Insert a row
CTRL+T	Insert a subtable
DEL	Delete a row
CTRL+G	Group the selected cells
CTRL+D	Set table defaults
CTRL+F	Find an item in the table
CTRL+B	Set table filename
ENTER	Complete a cell entry and select the cell below
TAB	Complete a cell entry and select the next cell to the right
SHIFT+TAB	Complete a cell entry and select the previous cell to the left
ESC	Cancel a cell entry

Working in Different Modes

Modes enable you to specify different conditions under which you can set different values for various fields of registers, memory maps, etc. For example, consider a scenario in which you want to define a start address value for the target element in a memory map in such a way so that the start address changes automatically depending upon the value of a Tcl variable being set. For such a case, you can define a mode similar to the one given below:

```
-start {=($var == 1) ? 10 : 20} -register reg0 -memory_map map
```

The above example defines a mode by using a Tcl variable, var. If the value of var is set to 1, the start address of the target element, reg0, is assigned value 10, otherwise 20. Now, consider that you set the value of var to 3, as shown below:

```
set var 3
```

On execution of the above command, GenSys automatically sets the start address of the target element, reg0, to 20.

Like Tcl variables, you can also define modes by using parameters, ports, and register bitfields. Consider the following example in which a mode is defined by using a parameter, param:

```
add_parameter -name param -type LOGIC -value 0001
add_register -name reg3 -offset 0 -width {= statement if {[p param] = =
1} { return 8 }else { return 16 }}
```

In the above example, a new parameter, param, is added and assigned a value, 1. In addition, a new mode is defined for the width of the register, reg3, such that the width is dependant upon the value of the parameter, param. If the value of param is set to 1, GenSys automatically sets the width of register, reg3, to 8 (as in this case). However, if param is assigned a value other than 1, GenSys sets the width of the register, reg3, to 16.

Note:

You can check the value currently stored in the parameter, param, by specifying the command, p param.

Similarly, you can define modes by using register bitfields. While defining a mode by using a register bitfield, specify the bitfield name in the format, *<register-name>.<bitfield-name>*. If the register of a bitfield belongs to a particular group, specify the bitfield name in the format, *<group-name>.<register-name>.<bitfield-name>*. Consider the following example:

```
add_register -name reg4 -offset 0 -width {= statement if {[bf reg0.b1] ==
11} { return 48} else { return 16 }}
```

The above example defines a mode in which the width of the register, reg4, is dependant upon the value of the bitfield, b1, of register reg0. Now, consider that you set the value of bitfield, b1, to 11, as shown below:

```
setbf reg0.b1 11
```

On execution of the above command, GenSys automatically sets the width of register, r4, to 48.

Note:

You can check the value currently stored in the bitfield, b1, by specifying the command, bf reg0.b1.

Using Procedures while Working with Modes

GenSys also allows you to use procedures while defining modes. Consider the following example in which you create a procedure, isParam:

```
proc isParam {first}{  if {$first == 1} {     return 1  } else  {
    return 2  }}
```

Now, consider that you you define a mode in which the width of the register, reg5, is dependant upon the value of the Tcl variable, param2:

```
add_register -name reg5 -offset 0 -width {= statement switch -exact --
$param2 {0 { return 2} 1 { return 88} 2 { return 48} default { return
16}}}
```

Here, you can set the value of param2 by passing the procedure name and its value, as shown below:

```
set param2 [isParam 1]
```

The above command sets the value of param2 to 1. This automatically updates the width of the register, reg5, to 88.

9

Customizing GenSys

You can customize GenSys by Creating Custom Generators in Perl or Tcl, Creating Custom Reports, and Modifying SCR Checks.

Creating Custom Generators

GenSys has some default generators that produce certain outputs. For example, Verilog and VHDL generators that generate the Verilog and VHDL files for a design.

You create your own generators in Perl or Tcl to produce customized output based on your requirement.

For details on creating custom generators, refer to the *Creating Custom Generators Using GenSys Objects* topic of *GenSys Customization Guide*.

Creating Custom Reports

You can create customized reports and add them in the tool parallel to the default GenSys reports.

For details on creating customized reports, refer to the *Creating Custom Reports* topic of *GenSys Customization Guide*.

Modifying SCR Checks

You can enable or disable SCR checks and modify their severities by using the Setup Rules dialog.

Encrypting Perl and Tcl Files

Encrypting Perl and Tcl files enable you to create encrypted Perl generators and Tcl procedures.

To encrypt such files, specify the following command while invoking GenSys:

```
gensys -encrypt -file <file-name>
```

For example, the following command encrypts a Perl file:

gensys -encrypt -file input.pl

Note the following points:

- To encrypt multiple files in one go, specify the files names to the -file *<file-names>* argument.
- You must specify the extension of the Perl (.pl) and/or Tcl (.tcl) file being encrypted. If you specify any other extension, GenSys reports an error.

- When you specify the -encrypt and -file arguments, GenSys ignores any other command specified on command line.

Files Generated After Encryption

The encrypted files are generated in the current working directory.

Based on the input file (.pl or .tcl), the following encrypted files are generated:

- input-OEM-gen_encrypt.genpl if the file name is input.pl
- input-OEM-gen_encrypt.gentcl if the file name is input.tcl

Using Encrypted Files - Decryption

This section covers the following topics:

- Using Encrypted Perl Files
- Using Encrypted Tcl Files

Using Encrypted Perl Files

Use the genesisapis::genRequire('<file-path'>) routine to use a procedure of an encrypted Perl file.

The following steps show the example of using an encrypted Perl file:

1. Create a Perl procedure to be encrypted. For example, consider the following procedure in the test.pl file:

   ```
   sub abc{    print "Inside sub... abc\n";}
   ```

2. Encrypt the test.pl file by using the following command:

   ```
   gensys -encrypt -file test.pl
   ```

   The above command creates the encrypted file test-OEM-gen_encrypt.genpl in the current working directory.

3. Add the following snippet to a Perl file that you can later include by using the -I command-line option:

   ```
   if(-f '/test/test-OEM-gen_encrypt.genpl'){  genesisapis::genRequire('/
   test/test-OEM-gen_encrypt.genpl');}
   ```

   Note:

   If an encrypted Perl file requires another encrypted Perl file, use genesisapis::genRequire in the first file to get the second encrypted Perl file.

4. Register the generator of the Perl file by using the RegisterGenerator("ABC","::abc") command.

   Note:
   The files intended to be encrypted should not contain Perl APIs of the name "dirname" or any such APIs that can be a part of other Perl packages, which might be doing something similar.

Using Encrypted Tcl Files

Use the gensys_source Tcl command to source the encrypted Tcl file so that you can run the Tcl procedure from that file.

The following steps show the example of using an encrypted Tcl file:

1. Create a Tcl procedure.

   For example, consider the following Tcl procedure in test.tcl:

   ```
   proc p_test {} {    puts "In the p_test procedure\n"}
   ```

2. Encrypt the test.tcl file by using the following command:

   ```
   gensys -encrypt -file test.tcl
   ```
   The above command creates the encrypted file test-OEM-gen_encrypt.gentcl in the current working directory.

3. Source the encrypted file in GenSys by using the following command:

   ```
   gensys_source test-OEM-gen_encrypt.gentcl
   ```
   Note:
   If an encrypted Tcl file sources another encrypted Tcl file, use gensys_source in the first Tcl file to source the second encrypted Tcl file.

4. Run the p_test procedure.

Note that you cannot run an encrypted Tcl file by using the -tcl command. Use gensys_source to include an encrypted Tcl file in a non-encrypted Tcl file, and then pass the non-encrypted Tcl file to the -tcl command.

10

Spreadsheet Methodology in GenSys Assembly

The spreadsheet methodology enables you to specify IP configurations and connectivity data in MS Excel. Once you populate the spreadsheet with data, the design teams can review the document for correctness. Therefore, this single document can serve as the golden database for connectivity information.

After review, you can import the data of the spreadsheet in GenSys and make required changes through GenSys GUI.

Once you make the changes in GUI, you can save the changes back to the spreadsheet.

The following points summarize the flow of this methodology:

1. Use GenSys RTL import methodology to import IPs and .lib cells into GenSys.
2. Use MS Excel write mechanism to create the Spreadsheet worksheet which has all these IP and .lib information.
3. Use the MS Excel spreadsheet to instantiate these IPs, map physical ports to interfaces, make connections and create hierarchies.
4. Re-import this MS Excel spreadsheet to GenSys and use the various GenSys widgets and debug reports to verify and validate the connectivity.
5. Use GenSys TCL commands and widgets to make some incremental modification to the SOC as needed.

6. Use GenSys RTL Generation capability to generate the RTL for the hierarchical SOC.

The Template MS Excel File

You must use the predefined MS Excel template file shipped with the GenSys release. Copy the following file at your local path:

SPYGLASS_HOME/examples/sdi/ss_template.xml

This template file is in the MS Excel 2003 XML format.

After copying this template file, you can rename the file and specify details to:

- Create components.
- Add ports and their properties.
- Add parameters.
- Add module instances and pass parameter override information to instances.
- Specify interfaces definitions.
- Map component ports to interface logical ports.
- Make adhoc, interface, and tieoff connections at the SoC level.
- Move instances in different hierarchies.

You must enter information in the spreadsheet under appropriate tabs. For details, see Worksheets In the Template Spreadsheet File.

Note:
Ensure that you do not change the existing content, such as headers and comments of the template file. You should only add rows under relevant under tabs to specify design information.

Importing Data From MS Excel File in GenSys

Once you update The Template MS Excel File with design information, import the spreadsheet information in GenSys by using the following Tcl command:

```
import_data -file <MS-Excel-File-Name> -type excel
```

During import, GenSys internally converts the spreadsheet data into appropriate Tcl commands, and these commands are then read in GenSys.

After importing, you can make changes to the SoC data (imported through the spreadsheet) in GenSys GUI through the Canvas View and various GenSys widgets.

After making changes, you can write back the changes in the spreadsheet file. For details, see Exporting Data from GenSys in the MS Excel File.

Exporting Data from GenSys in the MS Excel File

To save the changes made through GenSys GUI back to the spreadsheet file, use the following Tcl command:

```
export_data -as excel -file <Input-MS-Excel-File-Name> -out
<Output-MS-Excel-File-Name>
```

Worksheets In the Template Spreadsheet File

The template spreadsheet contains the following worksheets:

- The README Worksheet
- The TopDesignVLNV Worksheet
- The Instances Worksheet
- The Connections Worksheet
- The Components Worksheet
- The Interface Worksheet
- The InterfaceMapping Worksheet

Specifying Comments in Worksheets

The worksheet under each tab contains certain comments that are used by GenSys for internal purpose. Do not modify these comments. However, you can add any number of additional comments for your reference.

Specifying Rows to Ignore in a Worksheet

If you want to comment out any row that you do not want GenSys to consider while importing the worksheet, specify a hash (#) in any column, except the Description column, of a row.

For example, the second row in the following worksheet is commented out:

Worksheet Description:This sheet is used to specify the IP Instantiations, their parameter overrides (if any) and container hierarchical instance name.							
Vendor	Library	Name	Version	Instance Name	Parameters	Instance Name (Master Name)	Descripti on
uart_ven	uart_lib	uart	uart_2.3	u0	SIZE=>15; WIDTH=>RRR + 12		
	#	#pio					
		gpio		g0			
		gpio		g1_hier		I1/I2/I3	
		timer		timer128			
		timer		t0			
		sub_design		sub1			
		TTT		t1			
		GENSYS_AND		and_1			

The README Worksheet

This worksheet is not imported into GenSys. Any data added to this worksheet is for user's purpose to maintain project-specific information for easy review amongst various team members.

This worksheet contains a hidden comment that is internally used by GenSys to read information from other worksheets. If you unhide this comment, ensure that you do not modify the same.

However, you can add any number of comments for your reference.

The TopDesignVLNV Worksheet

This worksheet contains information on vendor, library, version, and name of the top-level design.

The following figure shows the example of this worksheet:

Worksheet Description:This sheet is used to specify the Top-level Name, Vendor, Library, Version and Description				
Vendor	Library	Name	Version	Description
ATRENTA	ATRLIB	mytop	2.0	New design being created

The details of each column of the above worksheet is described in the following table:

Column name	Description
Vendor	Vendor name of the top-level design
Library	Library name of the top-level design
Name	Name of the top-level design
Version	Version of the top-level design
Description	Description of the top-level design (Optional)

The Instances Worksheet

This worksheet contains details of IP instances instantiated in the top-level design.

The following figure shows the example of this worksheet:

Worksheet Description:This sheet is used to specify the IP Instantiations, their parameter overrides (if any) and container hierarchical instance name.							
Vendor	Library	Name	Version	Instance Name	Parameters	Instance Name (Master Name)	Description
uart_ven	uart_lib	uart	uart_2.3	u0	SIZE=>15; WIDTH=>RRR + 12		
	#	#pio					
		gpio		g0			
		gpio		g1_hier		I1/I2/I3	
		timer		timer128			
		timer		t0			
		sub_design		sub1			
		TTT		t1			
		GENSYS_AND		and_1			

The details of each column of the above worksheet is described in the following table:

Column name	Description
Vendor	Vendor name of the component/IP
Library	Library name of the component/IP
Name	Name of the component/IP
Version	Version of the component/IP
Instance Name	Name of the instance of the component/IP
Parameters	Semicolon-separated list of instance parameters. For details, see Specifying Parameters to be Overridden.

Column name	Description
Instance Name (Master Name)	The hierarchy in which the instance should be placed. For details, see Specifying the Hierarchy to Place an Instance in the Instances Worksheet. For pre-defined glue, the number of input pins can be specified by separating the instance pin with the number of input pins required by a '@'. For example, and_1@5. The width of each pin can be specified by the parameter “width” which can be given as “width=>3” to create 4 bit pin(s).
Description	Description of the component/IP instance. (Optional)

Specifying Parameters to be Overridden

In the *Parameters* column of the *Instances* worksheet, specify a semicolon-separated list of instance parameters (name-value pairs) that are to be overridden.

The name and value of the parameter must be separated by the => character. For example, you can specify the following value in this columns:

```
SIZE=>5;WIDTH=>7
```

Note:

The parameter values can be any RTL expression or can be defined in terms of design parameters also. The design parameters are further defined in the Components worksheet, which can have the component view of the top-level design.

Specifying the Hierarchy to Place an Instance in the Instances Worksheet

In the *Container Instance Name (Master Name)* column of the *Instances* worksheet, specify the hierarchy in which the instance should be placed. Use forward slash (/) as the hierarchy separator.

For example, consider you specify the hierarchy as a0(m0)/a1(m1). This implies that the instance is pushed insider the hierarchy a0/a1 where m0 is the master for a0 and m1 is the master for a1.

If you do not specify any hierarchy in this column, the instance is placed directly in the top-level design.

The Connections Worksheet

This worksheet shows the details of adhoc connections, such as pin-to-pin, export, tieoffs, open, and top-level floating ports with default values (if any).

The following figure shows the example of this worksheet:

Worksheet Description:This sheet is used to specify the top-level port defintions, connections between IP level pins etc. For more details refer to the Appnote.

Source	Source Pin/Interface	Destination	Destination Pin/Interface	Direction	Interface	Netname	Description
mytop	ea	u0	a		N		
mytop	AAA			OUT	N		
mytop	BBB			OUT	N		
mytop	and_1			IN	N		
mytop	and_2			IN	N		
mytop	and_3			OUT	N		
mytop	ecoin[2:0]			IN	N		
mytop	pdata[10:1]		0	OUT	N		
mytop	tieoff_port[3:2]		s10	OUT	N		
mytop	dataout[31:0]		s1	OUT	N		
mytop	ecoout[2:0]			OUT			
mytop	ecoin	mytop	ecoout				
u0	b						
u0	c[1:0]	g0	x[1:0]			n0	COMMENT
	\'b10	g0	x[3:2]				
	s10	t0	haddr[1:0]				pin

While importing the Connections sheet, if the top-level design port is not present in the Components sheet, the tool exports that particular pin or interface.

Whenever an export command is executed internally, an info message is logged in the gensys.log file.

The details of each column of the above worksheet is described in the following table:

Column name	Description
Source	Specifies the name of the component instance that acts as the source of a connection

Column name	Description
Source Pin/Interface	Specifies the name of the pin including MSB/LSB or an interface on the source. MSB/LSB should be provided as "pin[msb:lsb]" or pin [lsb]. Note: If you specify a tieoff using a tick (‘), ensure to escape it by using a backslash. For eg. \‘b10.
Destination	Specifies the name of the component instance that acts as the destination of a connection
Destination Pin/ Interface	Specifies the name of the pin including MSB/LSB or an interface on the destination. MSB/LSB should be provided as "pin[msb:lsb]" or pin [lsb]. Note: If you specify a tieoff using a tick (‘), ensure to escape it by using a backslash. For eg. \‘b10.
Direction	Specifies the port direction for top-level floating ports to be created.
Interface	Specifies if the connection is an interface connection or an adhoc connection. If this column is empty or the value N is specified, it indicates an adhoc connection. If the value is Y, it indicates an interface connection.
Netname	Specifies the name of the net of a connection.
Description	Specifies the connection description.

The Components Worksheet

This worksheet provides component details, such as component name, ports on a component, direction of each port, alignment of each port, and component parameters.

The following figure shows the example of this worksheet:

Component Name	Port Name	Port Direction	Parameters	Align	LSB>MSB	Default	Vendor	Version	Library	HDL Library	HDL Package	HDL Port Type	Model Name	Port Description
	a	IN		MSB										port_desc a
	b[1:0]	IN		MSB										
	c[7:0]	OUT		MSB										
	[5:0]	OUT		MSB										
	[5:0]	OUT		MSB										
	[10:1]	OUT		MSB										
	ort[0:4	OUT	WIDTH=>2; SIZE=>5;AN 1=>23	MSB	Y									
	sb_p	OUT		MSB	Y									
uart	d	OUT		MSB										
				MSB			uart_ve	uart_2.3	uart_li	uart_hc	uart_hdl_			
	x[3:0]	IN		MSB										
	[5:0]	IN		MSB										
	y[1:0]	IN		MSB										
	[5:0]	IN		MSB										
	:0]	IN		MSB										
gpio	z[1:0]	OUT	WIDTH=>8	MSB										
				MSB			gpio_ve	gpio_ve	gpio_li					

When an interface is exported or new interface is created, it results in new ports being added to the Components sheet when such a design is exported to Excel using SDI.

The details of each column of the above worksheet is described in the following table:

Column name	Description
Component Name	Name of the component
Port Name	Name of the port on the component
Port Direction	Port direction
Parameters	Semicolon-separated list of component parameters. For details, see Specifying Parameters to be Overridden.
Align	Specifies the port alignment, such as MSB or LSB
LSB>MSB	Specifies Y if LSB>MSB for a component port. Else, specifies N.
Default	Specifies the default port value
Vendor	Vendor of the component

Column name	Description
Version	Version of the component
Library	Library of the component
HDL Library	HDL library of the component
HDL Package	HDL package of the component
HDL Port Type	Port type
Model Name	Model information (type name, LSB, MSB, and language) of the active port
Port Description	Component description (optional)

The Interface Worksheet

This worksheet provides details of interfaces. While importing this worksheet, only those interface definitions are imported that are not already present in the GenSys library browser.

The following figure shows the example of this worksheet:

Worksheet Description: This sheet is used to specify the Interfaces in the design																
VLNV					Ports						Direction			Presence		
Interface Name	Library	Vendor	Version	Description	Logical Name	LSB>MSB	Port Direction	Signal Type	Default Value	Port Description	Master Mode	Slave Mode	System Mode	Master Presence	Slave Presence	System Presence
TCM	TMS570	TI_AEC	1.0.0	AEC Interface	TCPARITYOUT[13:0]		IN	DATA		Parity for the write data	in	out		required		
					TCACCTYPE[2:0]		INOUT	DATA		Access Type	in	out		required		
EGTM	EGS570	EG_AEC	1.0.1	EGTM_NPORRST	MODULE_NPORRST		OUT	RST		Module Power on Reset from GTM	in	in		required		
					PP1		INOUT				in	inout				
					PP2		INOUT				in	inout				
					lsb_bit_port[0:10]	Y	INOUT				in	in				
					PP3		INOUT				in	inout				
					SYS_NRST		INOUT	RST		System Reset	in	out		illegal		

The details of each column of the above worksheet is described in the following table:

Column name	Description	
VLNV	Vendor, library, name, and version of the interface.	
Port	LogicalName	Logical name of the port
	LSB>MSB	Specifies Y if LSB>MSB. Else, specifies N.
	Port Direction	Specifies the port direction, such as IN, OUT, and INOUT.
	Signal Type	Specifies the signal type, such as DATA and RST.
	Default Value	Specifies the default port value in case the port is unmapped.
	Port Description	Specifies port description. (Optional)

Column name	Description	
Direction	Master Mode	Specifies the direction for SYSTEM interfaces. The direction is specified as a comma-separated list of values of system-group name and direction for that group. For example, *GROUP1(IN), GROUP2(OUT)*
	System Mode	Specifies the direction of the MASTER interface.
	Slave Mode	Specifies the direction of the SLAVE interface.
Presence	Master Presence	Specifies if the presence of MASTER interface is required or optional in the master mode. Possible values to this column are REQUIRED and OPTIONAL.
	System Presence	Specifies if the presence of SYSYEM interface is required or optional in the system mode. Possible values to this column are REQUIRED and OPTIONAL.
	Slave Presence	Specifies if the presence of SLAVE interface is required or optional in the slave mode. Possible values to this column are REQUIRED and OPTIONAL.

The InterfaceMapping Worksheet

This worksheet enables you to map component ports to interfaces, which are defined in the *Interfaces* worksheet.

The following figure shows the example of this worksheet:

Component Name	Component Vendor	Component Library	Component Version	Port Name	Vendor	Library	Name	Version	Instance	Mirror	Type (Master/Slave /System/ Monitor)	Logical Port
				TCMAdrs_a0		TMS57			DTCM	NO	MASTER	TCERROR
				ITCMAdrs	TI_AEC	0	TCM	1.0.0	ITCM	NO	MASTER	TCERROR
				ARADDR_a0								NPORRST
				AWADDR_a0	EG_AEC	EGS570	EGTM	1.0.1	EGTM_1	YES	SLAVE	SYS_NRST
				IPhyTCMSize								[63:60]
				KKK2		TMS57						[53:50]
				KKK1	TI_AEC	0	TCM	1.0.0	ITCM			[53:50]
				WDATA_a0								PP1
				uart_lsb_port								ort[0:10]
				RDATA_a0	EG_AEC	EGS570	EGTM	1.0.1	EGTM_1			PP2
				ITCMWData								TCEN0
				0								TCRETRY
				ITCMRData					ITCM			TCEN1
				TCMEn_a0								2:3]
				TCMWEn_a0					DTCM			Y
				ITCMEn		TMS57						[7:0]
uart	uart_ven	uart_lib	uart_2.3	ITCMWEn	TI_AEC	0	TCM	1.0.0	ITCM			TCWE
mytop	ATRENTA	ATRLIB	2.0	act_1	TI_AEC	0	TCM	1.0.0	ITCM	NO	MASTER	TCWE
				TCMWData_a0								0]
				TCMRData_a0		TMS57						OR
gpio	gpio_ven	gpio_lib	gpio_ver	0	TI_AEC	0	TCM	1.0.0	DTCM	NO		TCRETRY

The details of each column of the above worksheet is described in the following table:

Column name	Description
Component Name	Name of the component
Component Vendor	Vendor of the component
Component Library	Library of the component
Component Version	Version of the component
Port Name	Name of the Port
Vendor	Vendor of the interface
Library	Library of the interface
Name	Name of the interface

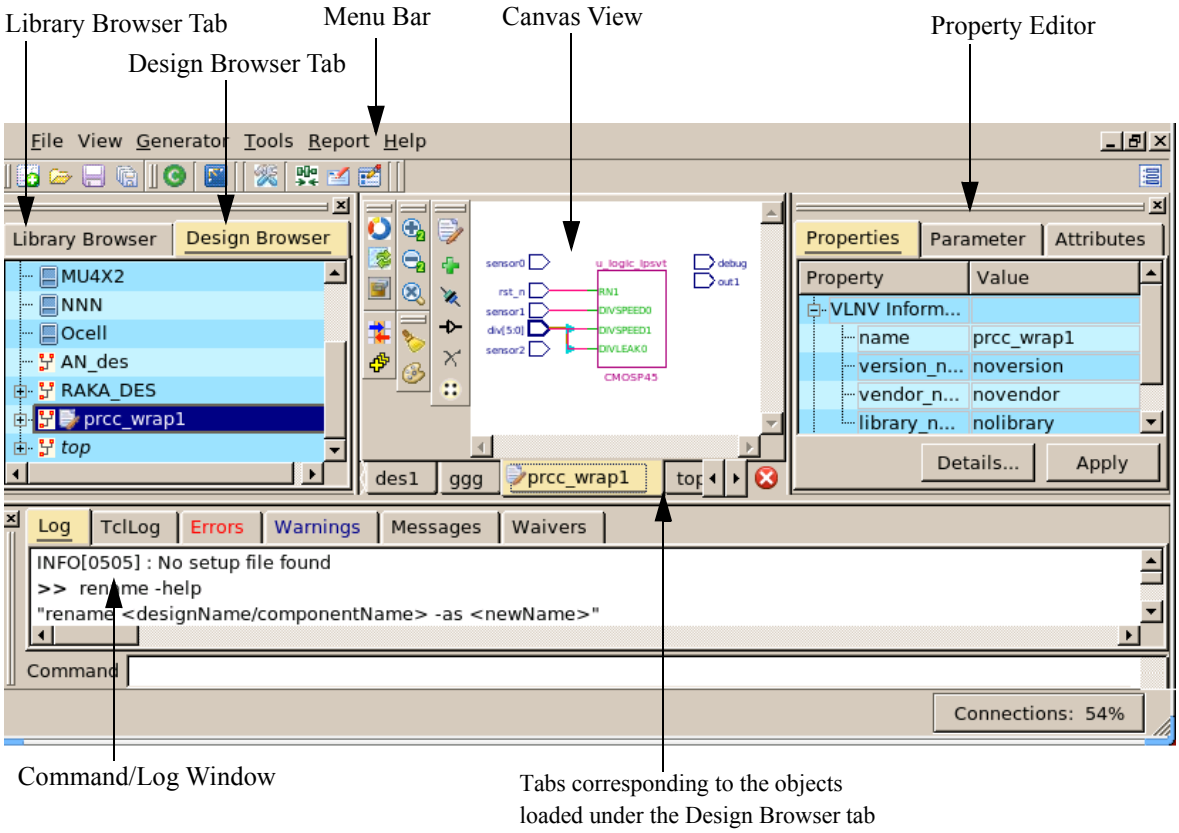
Column name	Description
Version	Version of the interface
Instance	Name of the interface instance
Mirror	Specifies if there is a need to instantiate the mirrored version of the interface
Type	Specifies if the interface instantiation is of the type master instantiation (MASTER), slave instantiation (SLAVE), system instantiation, or monitor instantiation
Logical Port	Name of the logical port

11

GenSys GUI Details

The GenSys GUI contains a menu bar, a toolbar, windows, and panes. This section describes the various panes and windows in GenSys.

Figure 11-1 GenSys Work Area



The GenSys work area consists of:

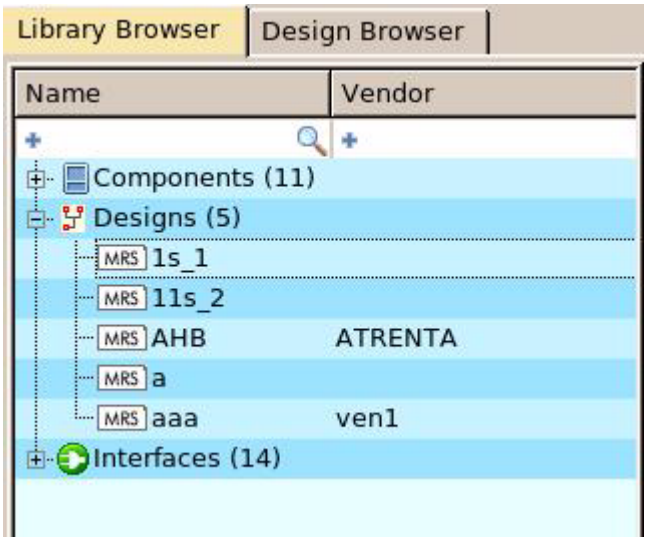
- The Library Browser Tab that displays all the previously loaded components, interfaces, and design. It also displays currently loaded design, components, or interfaces.
- The Design Browser Tab that displays currently loaded components, interfaces, or designs.
- The IP-XACT Viewer Pane displays the details of the underlying IP-XACT (XML) of the selected object
- The Command/Log Pane that displays message information generated by GenSys.
- The Modular Schematic Window that displays schematic of the selected design or component.
- The Incremental Schematic Window that displays the selected portions of a (flattened) design schematic across hierarchical boundaries.
- The Statistics Window that shows connection summary of a design in a graphical format.

- The Menu Bar that contains GenSys menu options.
- The Toolbar that contains icons for important functions.
- The canvas view in which you can update a design directly through schematic. For details, see Using the Canvas View.
- The property editor pane in which you can update basic properties of an object. For details, see Using the Property Editor Pane.

Library Browser Tab

Under this tab, GenSys displays the currently loaded components, designs, interfaces, and design configurations. GenSys also displays the number of each type of object at the object-type level, as shown in the following figure:

Figure 11-2 Library Browser Tab



Under the above tab, GenSys displays different type of icons against each object to identify the object type as MRS, Tcl, or IP-XACT. For example, in the above figure, the C1 component is an MRS object. Components, c and comp, are IP-XACT objects of IP-XACT versions 1.2 and 1.4, respectively.

The Library Browser tab also displays the VLNV information corresponding to each object, in the respective columns, such as Name, Version, and Library. By default, the Version and Library columns are hidden. To unhide them, right-click on the Library Browser header, select the Unhide option from the shortcut menu, and select Version and Library from the sub menu to show these columns. You can hide these columns in a similar way by selecting the Hide menu option.

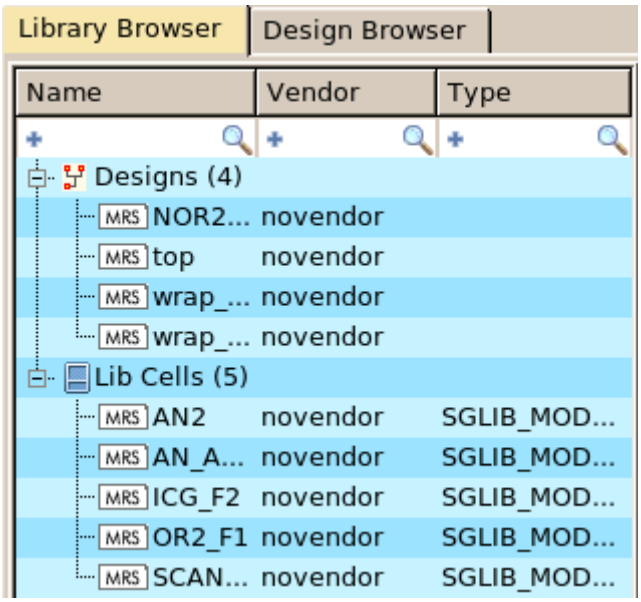
Note:
The Hide menu option is disabled for the Name column.

When you place the mouse pointer over an object in the Library Browser tab, GenSys displays a tool-tip that contains the object details such as VLNV and complete path and file name for that object.

Viewing Library Cells in the Library Browser

Library cells appear under the *Lib Cells* category in the *Library Browser*, as shown in the following figure:

Figure 11-3 Library Cells in the Library Browser



Library cells are always shown in the *Library Browser* irrespective of whether they are loaded in the design or not.

Filtering Objects in the Library Browser

You can filter objects in the Library Browser tab by using the filter bar, as shown in the following figure:

Figure 11-4 Filter-bar under the Library Browser Tab

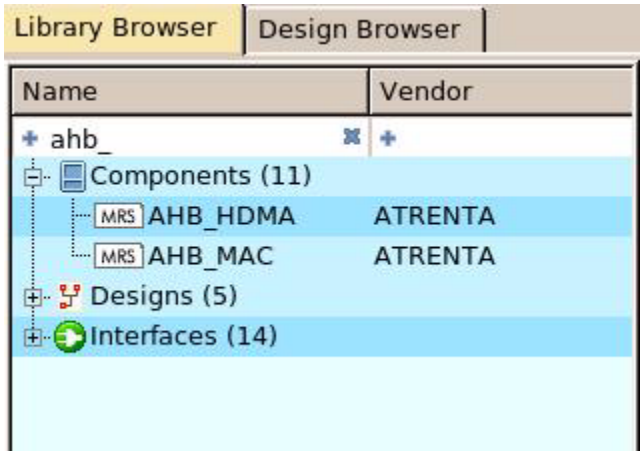
In the above filter bar, specify the filter criteria adjacent to the

button to display objects that match the specified filter criteria. As you start typing the filter criteria, GenSys automatically starts displaying filtered objects accordingly in the Library Browser.

For example, if you want to display only those objects whose names contain the string, ahb, specify that string adjacent to the

button in the Name column of the Library Browser tab, as shown in the following figure:

Figure 11-5 Filtered List Matching Filter Criteria



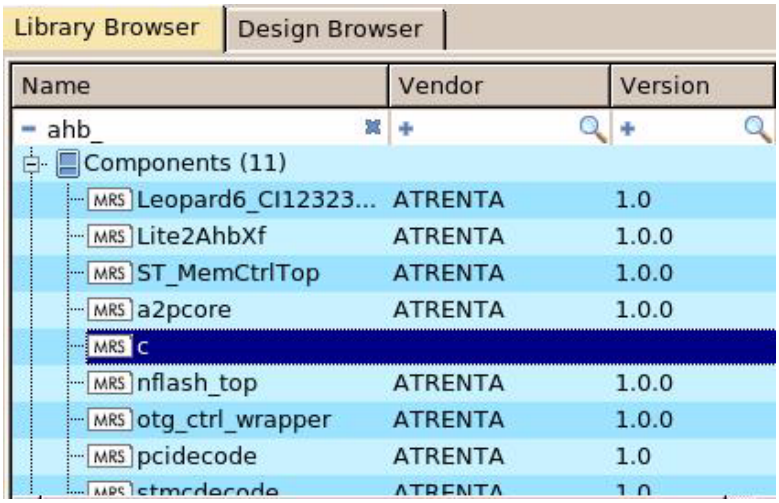
Conversely, if you want to display those objects whose names do not contain the string, bus, click the

button in the Name column. This changes the

button to the

button, and all those objects that do NOT match the specified filter criteria gets displayed, as shown in the following figure:

Figure 11-6 Filtered List Not Matching Filter Criteria



You can reset the filter by clicking the

button.

Library Browser Tab Menu Options

The following menu options appear when you right-click the Library Browser tab:

Add Lib File	Add/Remove Files With VLNV	Remove From Library
Edit Search Path	Scan Directory	Refresh Library
Replace IP	View Source File	Find .. Ctrl+F

Add Lib File

This menu option is used to add information of the .xml file to the Library Browser. This menu option is used to add a single .xml file to the library browser.

Add/Remove Files With VLNV

This menu option is used to add or remove files with their VLNV information from the Library Browser. In addition, this option is also used to load and dump the VLNV database file.

When you select this option, the Add/Remove Files With VLNV Information dialog appears. for details on this dialog, refer to the Add/Remove Files With VLNV topic.

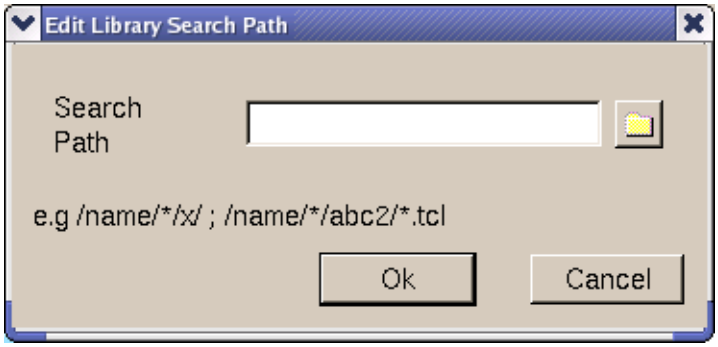
Remove From Library

This menu option is used to remove the information of the .xml file from the Library Browser. Choosing this option does not removes the file from the disk.

Edit Search Path

This menu option is used to specify the path of any .xml file of a component or an instance and include it in the Library Browser of the GUI. The following window opens when you select this menu option:

Figure 11-7 Edit Library Search Path



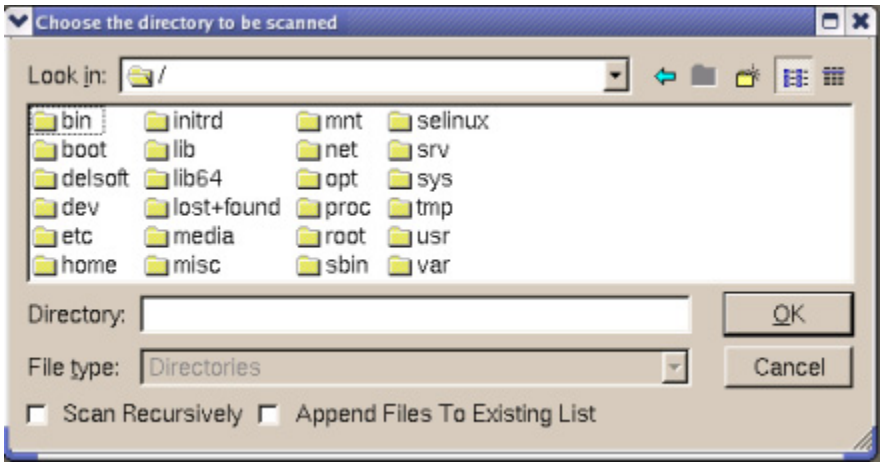
In the Search Path field, you can specify the directory path(s) of the file(s) that need to be included. If you need to specify more than one path, you can separate the paths using a semicolon (;).

Note that in the Search Path field, you can use regular expressions, such as *, ?, and so on, at any level of the file hierarchy.

Scan Directory

This menu option is used to populate the library browser with all the .xml files in the chosen directory. When you select this menu option, Choose the directory to be scanned dialog appears, as shown below:

Figure 11-8 Choose the directory to be scanned Dialog



Select the directory that needs to be scanned and click the OK button.

You can also select the Scan Recursively option to scan all the sub directories. The Append Files to Existing List option will append the files from the scanned directory to the Library Browser without clearing the previous contents of the browser.

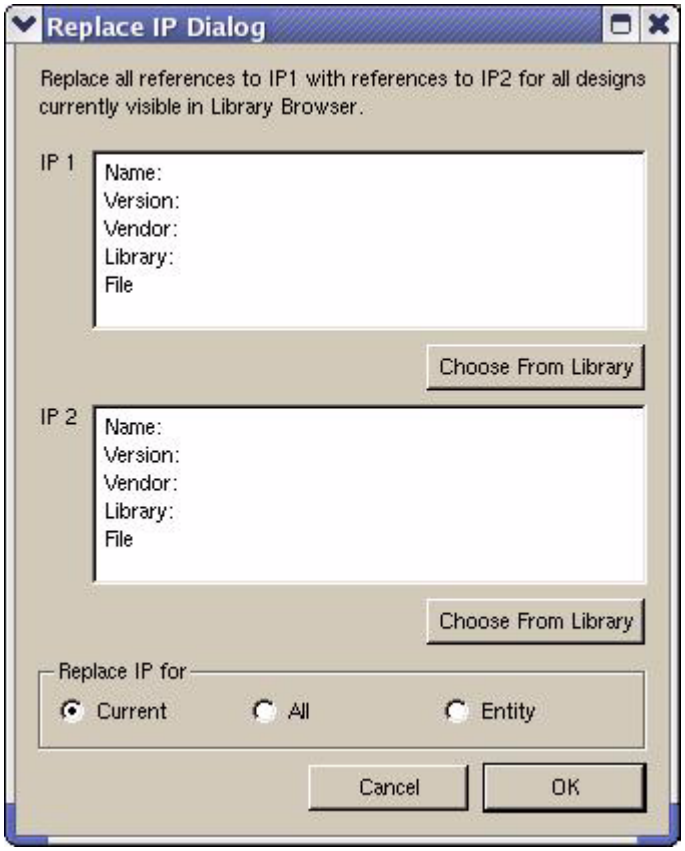
Refresh Library

This menu option is used to refresh all the entries in the Library Browser according to the last-modified time-stamp of the files. If a file corresponding to an entry has been overwritten after being added into the Library Browser, the file is loaded again and its corresponding object gets updated. If a file no longer exists, its corresponding entry is removed from the Library Browser.

Replace IP

This menu option is used to replace an IP with the new IP in the current or all the designs in the library or to update the currently active design itself. The following dialog box appears when you select this menu option:

Figure 11-9 Replace IP Dialog



Click Choose From Library next to the IP1 text box and choose an old IP, which you want to replace. Similarly, choose a new IP, with which you want to replace IP1, in the IP2 text box.

From the Replace IP for section, select any of the following options:

- Current: To replace an IP with a new IP in the current design. This option is selected by default.
- All: To replace an IP with a new IP in all the loaded designs.
- Entity: To replace a design with a new design. In this case, only the design is updated and the instances inside the design remain unchanged.

  While updating the design, the ports and interfaces on the current design are replaced with the ports and interfaces of the new master, if their names match. In addition, the connections between the design ports/interfaces and internal instances are also maintained wherever possible. The values of parameters are also maintained.

Click OK to replace the IP in the library.

Note:
Currently, this feature to provide port/interface/parameter mapping is available in GUI only. You cannot provide mapping in batch mode.

View Source File

This menu option is used to display the XML file, present on the disk, for the selected object.

When you select this menu option, GenSys displays the corresponding XML file of the selected object in a separate Text Viewer window.

Find .. Ctrl+F

This menu option is used to search for items that match the entered text. The following dialog box appears when you select this menu option:

Figure 11-10 Find: Library Dialog

Enter the item that needs to be searched in the Find text field. Select the options that matches your requirement and click the Find Next button.

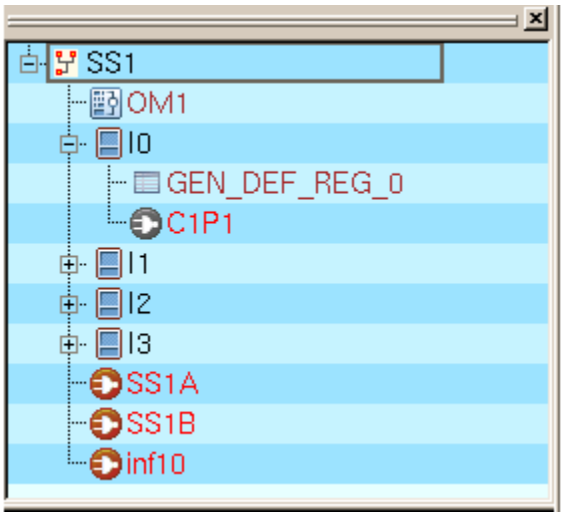
Design Browser Tab

The page under this tab is called Design Browser.

This page displays currently loaded design files, components, interfaces, or design configurations. It also displays the component registers, if any, under each component.

The following figure displays the currently loaded design in the Design Browser:

Figure 11-11 Design Browser Tab



Whenever a component/design/interface/design configuration is changed, its name gets displayed in italics. This indicates that the object has been modified and needs to be saved. If you close the updated component/design/interface/design configuration without saving it, GenSys prompts you to save that object. Similarly, if master of an instance under any design or component is changed, names of instances of that master are also shown in italics, indicating that they have been modified.

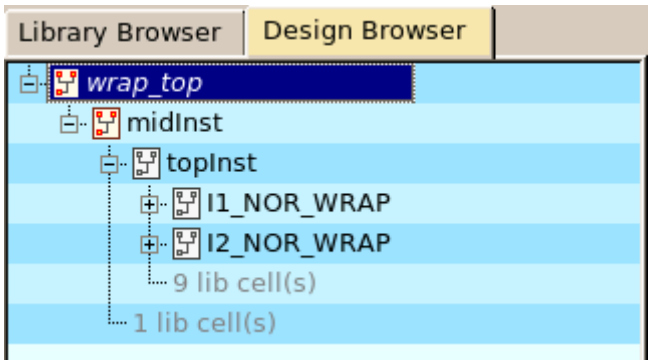
Note:

If you make changes in multiple objects and use the save Tcl command or select the File -> Save menu option to save the changes, GenSys saves the currently active object and prompts you to save all other objects in which you have made changes.

Library Cells in the Design Browser

If you have loaded library cells in GenSys, the Design Browser displays the number of library cells, as shown below:

Figure 11-12 Number of library cells shown in Design Browser



Coloring Convention for Instantiated Interfaces in Design Browser

You can instantiate interfaces in a design or in a component. In designs, you can define connections to interfaces as a whole or pins of interfaces. The following color coding is followed to indicate interface names shown under the Design Browser.

- Green color indicates that the interface is fully connected to the design.
- Red color indicates that the interface is not connected to design or component.
- Orange color indicates that the interface is partially connected to the design or component.
- Gray color indicates that the interface has some connections but system can not infer whether it is fully connected. This situation can happen during pre-elaboration only. Once elaboration is performed, the status of a gray interface would change to green/orange.

The colors of the interface names change when you edit or delete connections to that interface.

Design Browser Tab Menu Options

The following menu options appear when you right-click under the Design Browser tab:

Open Master	View XML	View RTL Source
Read Master Component	Reconfigure Tie off Connections	Reconfigure Instance
Update Masters	Delete	Load to IS

Add to IS	Find	Make Leaf Component
Group	Ungroup	

Open Master

This menu option is used to display the master of an instance.

When you select this menu option, the master appears in the Canvas view.

View XML

This menu option is used to display the XML file, present on the disk, for the selected object.

When you select this menu option, GenSys displays the corresponding XML file of the selected object in a separate Text Viewer window.

View RTL Source

This menu option is used to display the RTL of the selected object.

When you select this menu option, GenSys displays the corresponding RTL file of the selected object in a separate window.

You can edit the RTL as per your requirements.

Read Master Component

The Read Master Component menu option is used to read the master component data after loading an instance from the design view.

Reconfigure Tie off Connections

The Reconfigure Tie off Connections menu option is used to configure adhoc tie-off connections for an instance. When you select this menu option, the Configure TieOff Values dialog appears using which you can modify the tie-off value (per port) and create or remove tie-off connections. You can also specify the LSB, MSB, plane name, tie-off value, and alignment for each port.

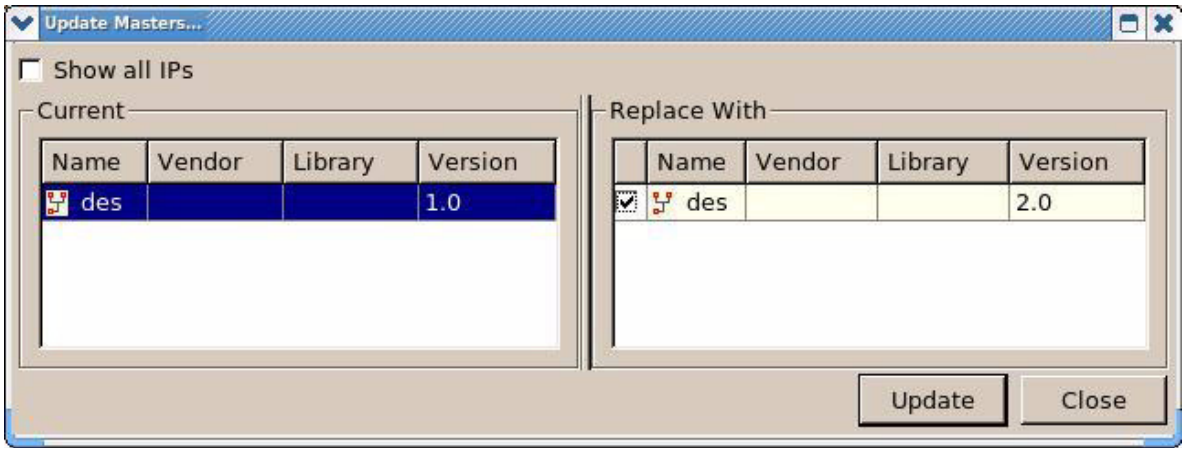
Reconfigure Instance

The Reconfigure Instance option enables you to configure the parameters of the selected instance in a design.

Update Masters

This menu option is used to upgrade the VNLV information of the IPs instantiated in the current design. When you select this menu option, the Update Masters dialog appears, as shown in the following figure:

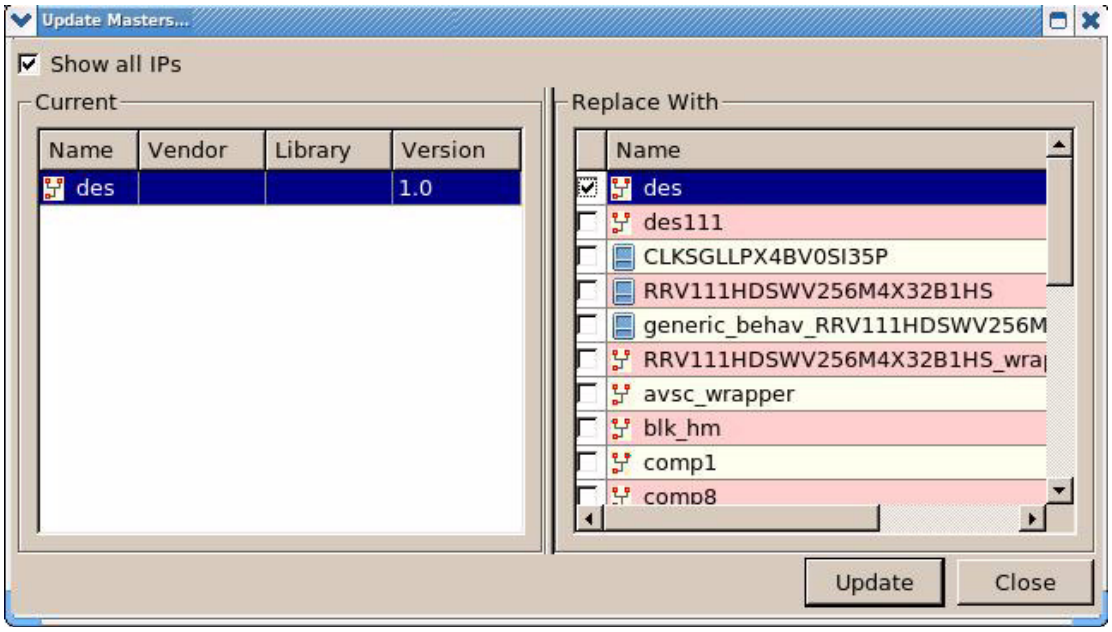
Figure 11-13 Update Masters Dialog



The above dialog, the Current section shows the master of the currently selected instance. The Replace with section lists the masters whose names match with the name of the master in the Current section.

Select the master name in the Replace with section to bind the VNLV information of that master with the VNLV information of the master appearing in the Current section.

If you want to list all the masters of the active design in the Current section, select the Show all IPs check box, as shown in the following figure:



Click the *Update* button to update the selected master with the selected VNLV.

Delete

This menu option is used delete the selected design, component, interface, or design configuration under the Design Browser.

Load to IS

This menu option is used to load the selected instance in the Incremental Schematic window. For details about the Incremental Schematic window, see Incremental Schematic Window section.

Add to IS

This menu option is used to add the selected instance in the Incremental Schematic window. For details about the Incremental Schematic window, see Incremental Schematic Window section.

Find

The Find... menu option is used to search for items that matched the entered text.

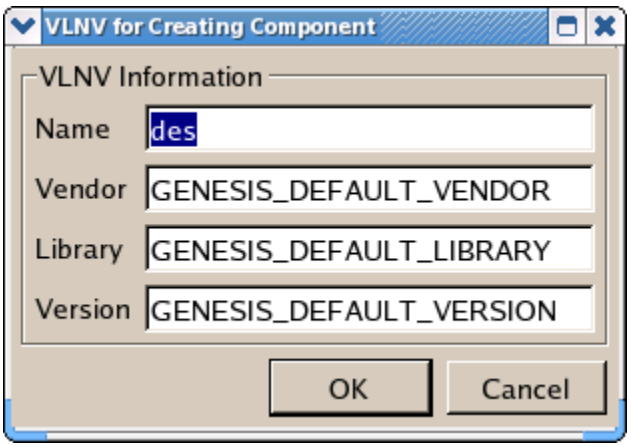
Make Leaf Component

This menu option is used to create a component view of a design instantiated under a top-level design.

Note:

This option does not appear if the selected design is a top-level design or if the top-level design is in the read mode.

When you select this menu option, GenSys prompts you to specify the VLNV details of the component through the VLNV for Creating Component dialog. This dialog is shown below:



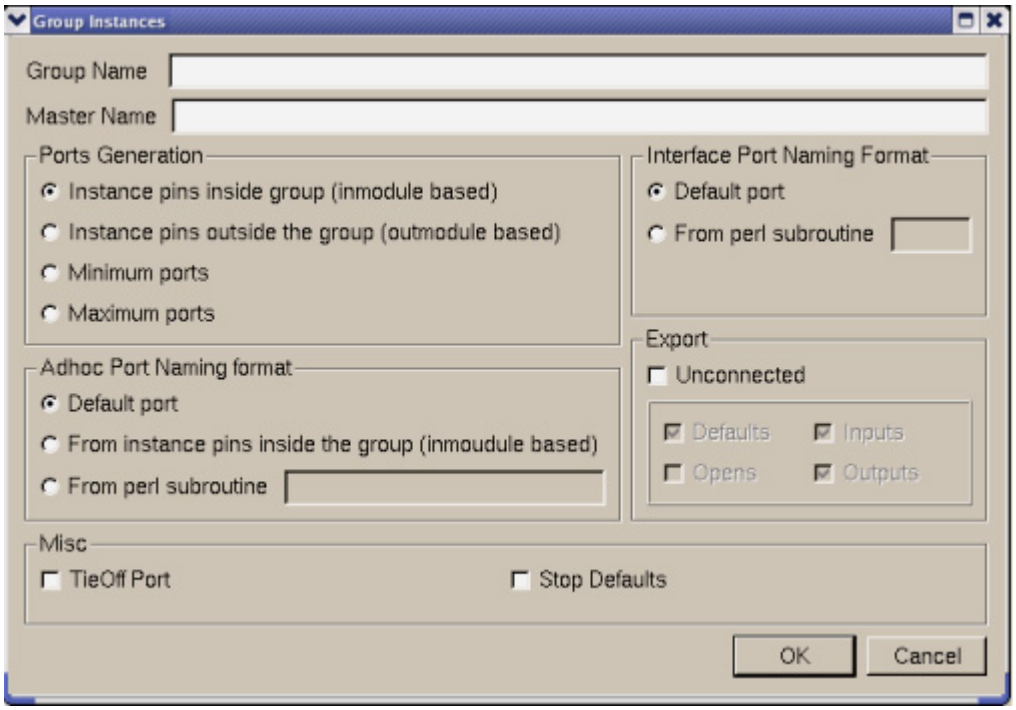
In the above dialog, when you click the OK button, GenSys converts the selected design to a component.

This component contains only design boundary information, such as ports on the design. It removes all the other information, such as instantiated sub-systems and connectivity that were present in that design.

Group

This menu option is used to group instances in a design into a new level of hierarchy. When you select instance(s) to be grouped and click on this menu option, the Group Instances dialog appears, as shown in the following figure:

Figure 11-14 Group Instances Dialog



The above dialog enables you to perform the following tasks:

- Specifying Group and Master Names
- Managing Number of Ports
- Specifying Adhoc Port Naming Convention
- Specifying Interface Port Naming Convention
- Exporting Unconnected Ports
- Specifying Miscellaneous Options

Specifying Group and Master Names

You can specify the name of the group to be created in the Group Name text box and the name of the master of the group/subsystem in the Master Name text box.

If you do not specify the master name, <group-name>_Dsgn is considered as the default master name for the group/subsystem, where *<group-name>* is the name of the group.

If the master name conflicts with an existing design name, the unique master name will be <group-name>_Dsgn(_<suffix>), where *<suffix>* is a numerical value starting with 0 that gets incremented by one.

Note:

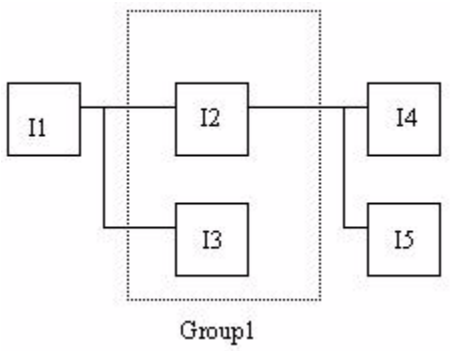
If the master name of the group/subsystem conflicts with the name of an already existing design or component, GenSys flags an error and does not proceed for grouping.

Managing Number of Ports

The number of ports that get created on a group depends on connections to/from the group. You can control the number of ports to be created on a group by using the following options in the Ports Generation section:

- Instance pins inside group (inmodule based)

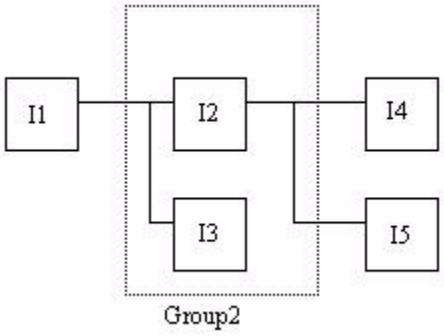
Select this option to create in-module-based connections while grouping. In this case, the number of ports on the group is equal to the number of distinct instance terminals inside the group, as shown in the following figure:



This option minimizes the number of ports if a driver is inside the group and maximizes the number of port if the driver is outside the group.

- Instance pins outside group (outmodule based)

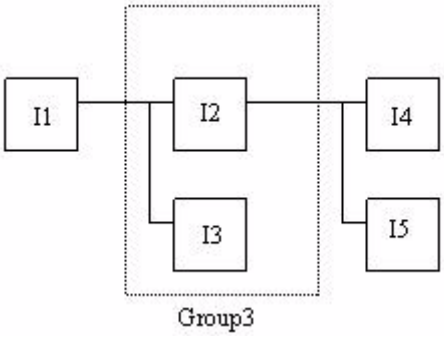
  Select this option to specify out-module-based connections while grouping. In this case, the number of ports on the group depends upon the number of distinct external ports connected with any instance terminal inside, as shown in the following figure:



Using this option minimizes the number of ports if a driver is outside the group and maximizes the number of ports if the driver is inside the group.

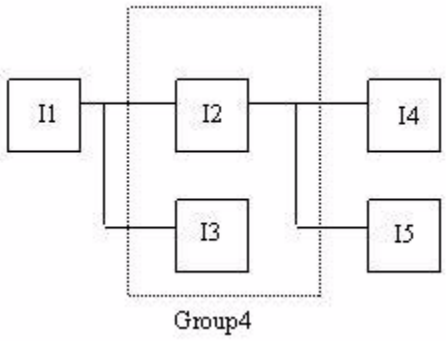
- Minimizing ports

  Select this option to create minimum number of ad-hoc ports on the group. In this case, ports get created based on the number of driver ports present (both external and internal to the group boundary) in connectivity, as shown in the following figure:



- Maximizing ports

  Select this option to create maximum number of ad-hoc ports on the group. In this case, the number of ports depends upon the number of driven ports, which crosses the hierarchy, as shown in the following figure:



Note:
In case of interface connections, interfaces and their actual ports will always be in-module-based, irrespective of any option selected in the Gen Ports section.

Specifying Adhoc Port Naming Convention

You can specify the naming convention for ports by selecting one of the following options in the Adhoc Port Naming format section:

- Default port

  This is the default port naming convention in which a port is named as *<instance-name>_<pin-name>*.

  Here, *<instance-name>* and *<pin-name>* is derived from instances (and their respective pins) within or outside the group of an in-module or out-module based connection, respectively.

  If the port name conflicts with an existing port name, the unique port name, *<instance-name>_<pin-name>_<suffix>*, is generated, where *<suffix>* is a numerical value.

- From instance pins inside the group (inmodule based)

  This type of port naming convention enables you to name ports on the basis of instances inside the group. The naming convention of the port name is *<instance-name>_<pin-name>*.

  If the port name conflicts with an existing port name, the unique port name generated is *<instance-name>_<pin-name>_<suffix>*, where *<suffix>* is a numerical value.

  This type of naming convention is useful if you want to name ports based on instances inside the group, but specify out-module-based connections or create maximum/minimum number of ad-hoc ports on the group.

- From Perl subroutine

This type of port naming convention enables you to specify your own Perl function in the From perl subroutine text box that provides a default naming convention for the ports.

The Perl subroutine accepts the following arguments:

- First argument is the adhoc connection type-item pointer inside the group
- Second argument is an adhoc connection type-item pointer outside the group.

If you want to use alias names of instances to name ports on a group, you can provide alias names for instances as attributes inside the Perl function.

While creating minimum/maximum ports on a group, it is important to identify a driver port. However, identification of the driver port is not possible if one side of the connection contains an inout port. In such cases, ports are named on the basis of instance-port combination inside the group.

Note:
For IN-IN or OUT-OUT type of connections, a unique port gets created on the group.

Specifying Interface Port Naming Convention

Similar to the adhoc port naming convention, you can also name interface ports (logical port of an interface) on a group instance. You can specify a naming convention for the interface ports by selecting one of the following options in the Interface Port Naming Format section:

- Default port

  This is the default interface port naming convention. If an interface connectivity exists between an instance, which is being grouped with another instance outside the group, the name of the actual interface port exported on the group boundary will be *<instance-name>_<actual-port-name>*. Here, *<instance-name>* and *<actual-port-name>* are derived from the instances (and the actual names of their respective ports) inside the group.

- From Perl subroutine

  This type of port naming convention enables you to specify your own Perl function in the From perl subroutine text box that provides the default naming convention for the interface ports. The Perl subroutine accepts two arguments. First is the interface connection type-item pointer inside the group and second is the interface connection type-item pointer outside the group.

  This Perl function subroutine enables you to specify the interface port naming convention in the following ways:

  - By specifying prefix/postfix

  You can specify the actual names of the interface ports exported on the group boundary as *<prefix>_<lport-name>* or *<lport-name>_<postfix>*. Here, *<lport-name>* refers to the

logical port name of interface instance inside the group, and *<prefix>* / *<postfix>* are the strings specified in the Perl subroutine function.

In this case, the Perl subroutine should return the value -prefix [*<prefix>*_] for prefix and -postfix [_*<postfix>*] for postfix. If the Perl subroutine returns both these values, the name of the interface port exported on the group boundary will be *<prefix>*_*<lport-name>*_*<postfix>*.

- By specifying aprefix/apostfix

You can specify the actual names of the interface ports exported on the group boundary as *<prefix>*_*<aport-name>* or *<aport-name>*_*<postfix>*. Here, *<aport-name>* refers to the actual port name of interface instance inside the group. *<prefix>* and *<postfix>* are the strings specified in the Perl subroutine function.

In this case, the Perl subroutine should return the value -aprefix [*<aprefix>*_] for prefix and -apostfix [_*<apostfix>*] for postfix. If the Perl subroutine returns both these values, the name of the interface port exported on the group boundary will be *<aprefix>*_*<lport-name>*_*<apostfix>*.

- By mapping logical port names with the actual port names

In this case, the Perl subroutine returns a comma-separated list of logical port name and actual port name pairs, as shown in the following example:

-map [L1:A1,L2:A2,L3:A3].

Here, logical port names of interface instances are mapped with the actual port names, and the interface ports created on the group boundary have the same name as the actual port name specified in the mapping.

You should specify a mapping for all the logical port names with the actual port names. If you do not provide mapping for all logical ports then logical names will be used to create the actual names for the rest of the logical ports.

Please note that if you specify all the above naming conventions together, priority for a naming convention will be given in the following order:

```
map ->  -aprefix/-apostfix -> -prefix/-postfix
```

Exporting Unconnected Ports

You can export the unconnected terminals of the instances being grouped to the group boundary by selecting the Unconnected check box. By default, this check box is not selected. When you select this check box, the following check boxes gets enabled:

- Defaults

  Select this check box to export ports with default connections (default tieoff or default open) to the group boundary. By default, this check box is selected.

- Inputs

Select this check box to export the unconnected input ports to the subsystem boundary. By default, this check box is selected.

- Opens

  Select this check box to export all the intentional open ports to the subsystem boundary. By default, this check box is not selected.

- Outputs

  Select this check box to export the unconnected open output and inout ports to the subsystem boundary. By default, this check box is selected.

Specifying Miscellaneous Options

The Group Instances provides a Misc section that contains the following check boxes:

- TieOffPort

  Select this check box to specify that ports should be generated on the group instance for the tie-off connections inside the group. By default, this check box is not selected.

- Stop Defaults

  Select this check box to propagate the default values (tieoff or open) of unconnected ports of instances inside the group to the group boundary. By default, this check box is not selected.

  Note:
  Currently, logical connections (nets) are not handled during grouping.

For more details, refer to The Group/Ungroup Feature.

Ungroup

This menu option is used to ungroup instances that are grouped in a design and brings them under the upper level directly under the top design. This right-click option is enabled only on single selection of top-level instance of a design, which should be a subsystem. Connections are also recreated accordingly.

Note:
Currently, logical connections (nets) are not handled during ungroup.

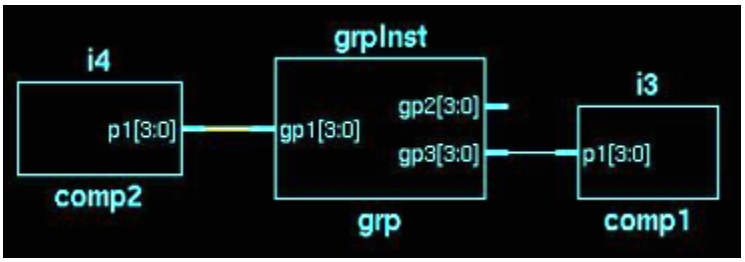
While ungrouping, if any open connection is present on the ports of the design being ungrouped, GenSys creates a dummy buffer instance corresponding to any such port on the group subsystem. The output port of this buffer instance then preserves all the connections inside the group boundary. This technique prevents any loss of connectivity information and maintains logical equivalence.

Each buffer instance is wrapped within COM_RTL_BUF where each out[N:0] is connected to in[N:0] (one-to-one mapping) as glue. The buffer instances with same widths share the same master with the name UNGROUP_GENERATED_COMP_<*width*>.

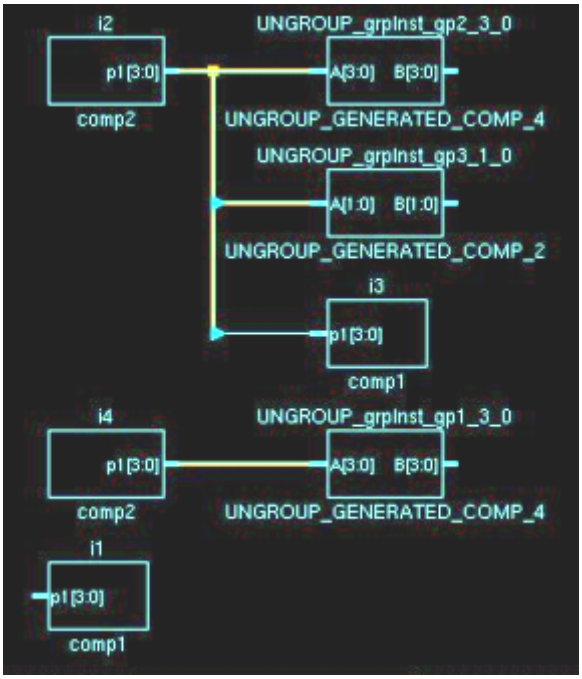
Each generated buffer instance has a unique name in the following format:

UNGROUP_<*group-name*>_<*port-name*>_<*msb*>_<*lsb*>

For example, consider the following figure representing the Modular Schematic before ungrouping:



After ungrouping, the Modular Schematic appears as shown below:



Here, note that the buffer instances, UNGROUP_grpInst_gp1_3_0 and UNGROUP_grpInst_gp2_3_0, each of width 4, share the same master, UNGROUP_GENERATED_COMP4.

Note:

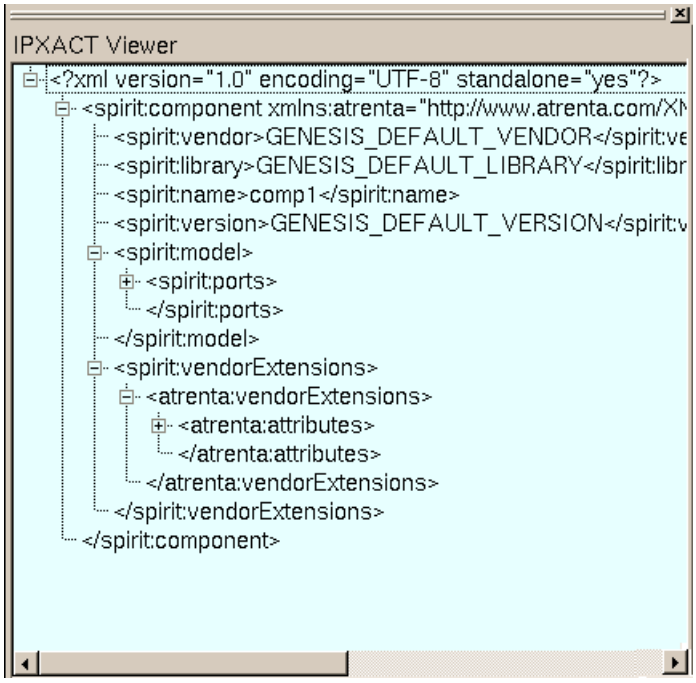
Currently, GenSys does not support interface open connections and interface loop-back connections while ungrouping.

For more details, refer to The Group/Ungroup Feature.

IP-XACT Viewer Pane

The IP-XACT Viewer pane displays the details of the underlying IP-XACT (XML) of the selected object, as shown in the following figure:

Figure 11-15 IP-XACT Viewer Pane



You can load the details of an IP-XACT object in the IP-XACT Viewer pane by double-clicking the required IP-XACT object appearing in the Library Browser tab.

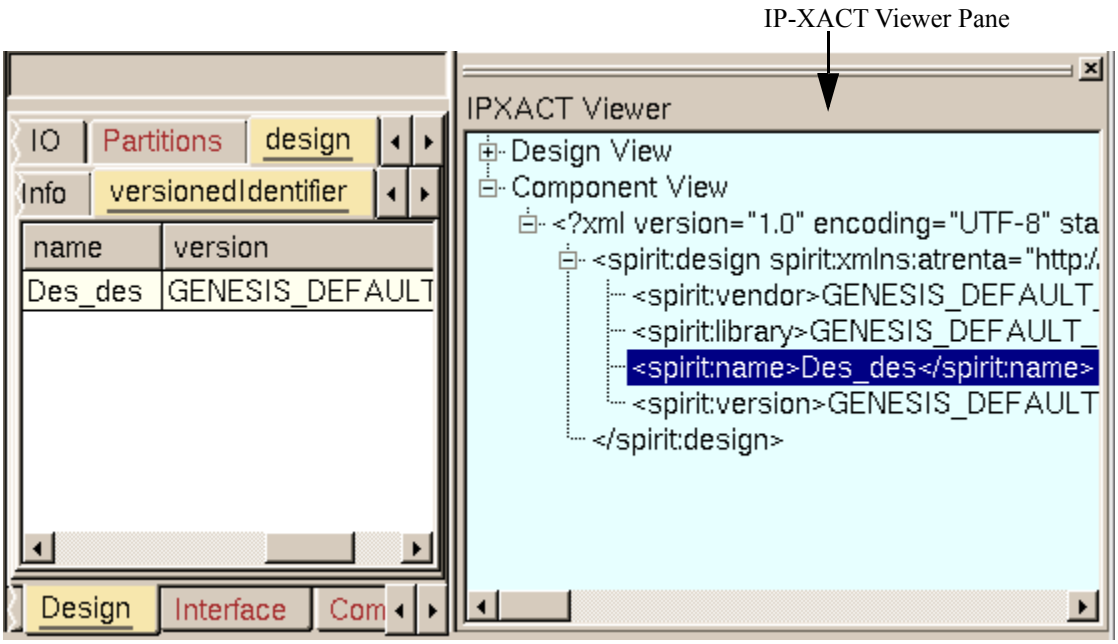
Note:

If you try to open the IP-XACT Viewer pane, which was originally closed while loading a design, GenSys prompts you to reload the design.

You can also cross-probe between the IP-XACT XML tree (appearing in the IP-XACT Viewer pane) and the corresponding IP-XACT table by double-clicking the required child element in

the IP-XACT XML tree. When you double-click on the required child element in the IP-XACT Viewer pane, GenSys takes you to the corresponding IP-XACT table column. For example, consider the scenario illustrated in the following figure:

Figure 11-16 IP-XACT Viewer Pane in GenSys Window



In the above figure, when you double-click the child element, name, in the IP-XACT Viewer pane, GenSys cross-probes to the name column of the corresponding IP-XACT table. Now, when you select any value in the vendor, version, or library column of the IP-XACT table, GenSys cross-probes to the corresponding child element in the IP-XACT Viewer pane.

You can also edit values in the IP-XACT tables. Once you edit the required value (say change Des_des to Des1 in the above figure), click the

button in the toolbar to reflect that change in the IPXACT Viewer pane.

Showing/Hiding the IP-XACT Viewer Pane

You can display the IP-XACT Viewer pane in GUI by performing any of the following actions:

- Select the IP-XACT Viewer option from the View->Views menu.

- Right-click on the toolbar, and select the IP-XACT Viewer option from the short-cut menu.
- Run the following Tcl command:

```
set_ipxact_viewer_mode on
```

On performing any of the above actions, GenSys displays the IP-XACT Viewer pane on the right-side in the GUI window.

To close the IP-XACT Viewer pane, perform any of the following actions:

- Deselect the IP-XACT Viewer option from the View->Views menu or from the short-cut menu of the toolbar.
- Run the following Tcl command:

```
set_ipxact_viewer_mode off
```

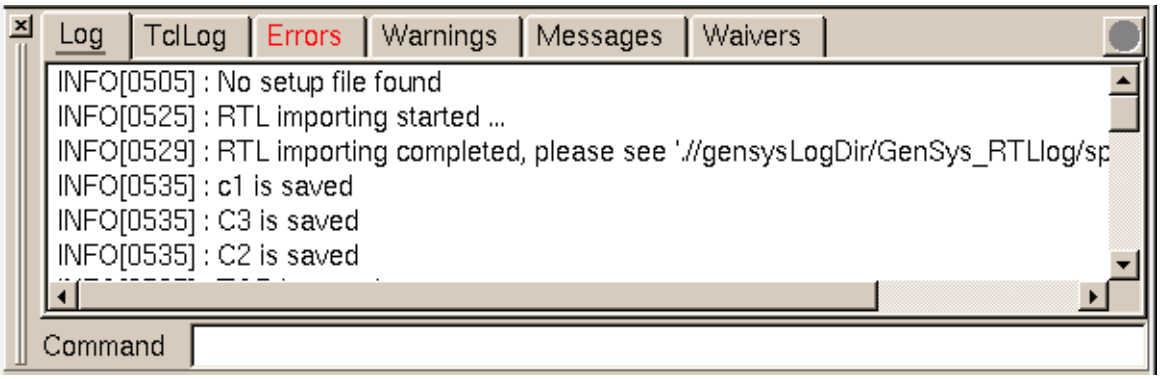
Note:

When you close GenSys window, the display status of the IP-XACT Viewer pane (whether shown or hidden) is saved in the gensysrc file. Next time when you load GenSys again, the IP-XACT Viewer pane is shown or hidden depending on the display status saved in the .gensysrc file.

Command/Log Pane

The Command/Log Window displays various messages whenever you perform any action in the GUI. Such messages include processing information or any error/warning messages related to the task being performed or a task completed.

The following figure shows the Command/Log Window:



The Command/Log Window contains various tabs as described in the following table:

Tab	Purpose
Log	Displays the log or processing messages corresponding to the tasks performed in the Command text box.
TclLog	Displays the Tcl commands corresponding to the actions performed in the GUI, such as selecting tables, adding ports, and so on. Also, displays the Tcl commands specified in the Command text box. Refer to the Specifying Commands section. A sample tcl log follows: gentclcmd.log ----------# New GenSys Session # Thu Jul 26 16:45:03 2012 run ".../gensys/source/SPYGLASS_HOME/.gensys. config" run ".../.gensys.setup" run ".../case1/test.tcl" mod_sch_show -design des elaborate
ERROR	Displays error messages generated due to the any processing of Tcl commands given in the Command text box. To close this tab, click . If any error message is present under this tab, the text of this tab appears in red color.
WARNING	Displays warning messages generated due to the any processing of Tcl commands given in the Command text box. To close this tab, click If any warning message is present under this tab, the text of this tab appears in blue color.
Messages	Displays various messages under a message tree.
Waivers	Displays the waived message.

When you right-click on any tabs in the Command/Log Window, following menu options appear:

Menu Options	Purpose
Find	Finds a text string in the Command/Log Window.
Copy	Copies a string from the Command/Log Window.
Clear	Clears the text in the Command/Log Window.
Select All	Selects all the items in the Command/Log Window.
Log Filter	(Appears only in the Log tab window). Adds filters on the messages appearing in the Log tab.

Viewing Messages in Different Formats

You can view messages displayed under the Messages tab in a tree-view or a flat view format. To select the required format, click the Message Filters icon (

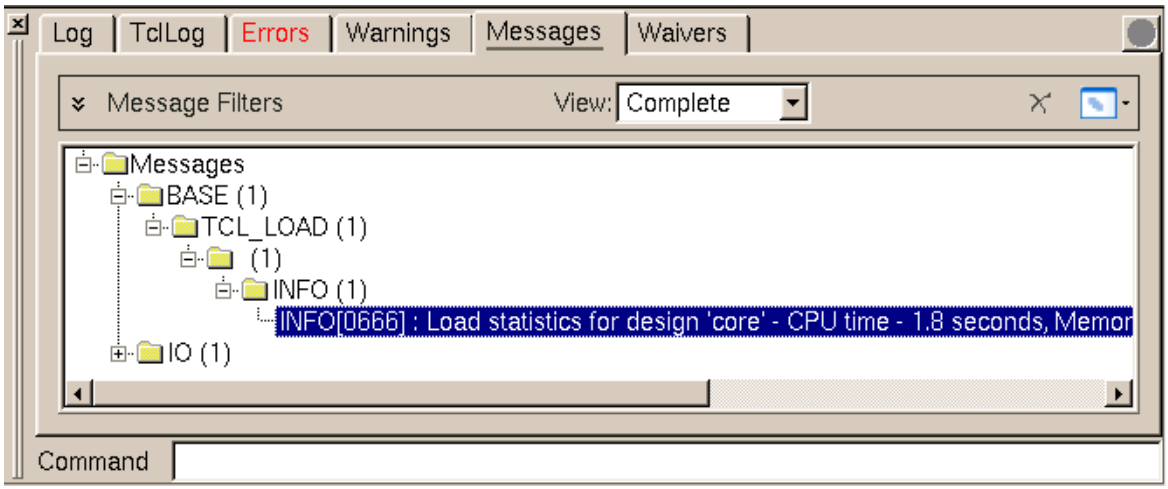
) and select the required option (Tree View or Flat View) from the displayed list.

Viewing Messages in a Tree View Format

In this format, messages are displayed in a hierarchical tree-like structure. Each message in this structure appears under an appropriate severity node, such as ERROR, WARNING, or INFO.

The following figure shows messages displayed in a tree view format:

Figure 11-17 Tree View Format

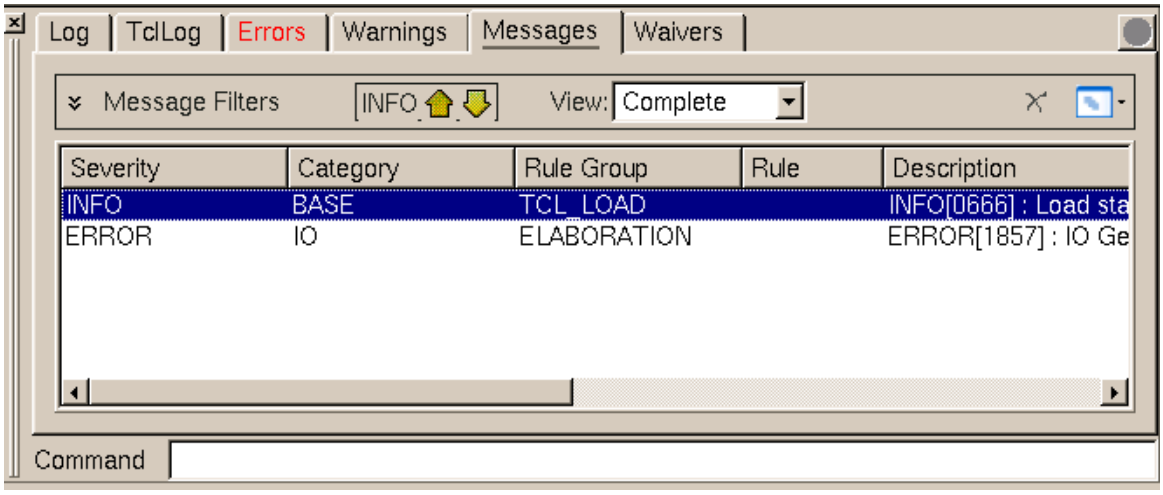


Viewing Messages in a Flat View Format

In this format, all messages appear at a flat-level, that is, without any hierarchy.

The following figure shows messages displayed in a flat view format:

Figure 11-18 Flat View Format



In this view, you can use the navigator buttons (

) to navigate between the messages of the selected severity.

Viewing Messages in Different Modes

You can view messages in any of the following modes:

- Complete Mode

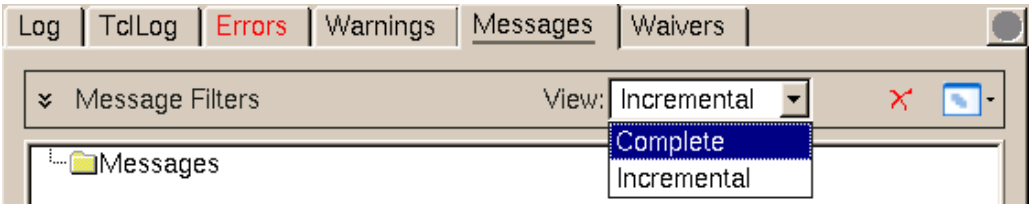
  In this mode, GenSys displays all violation messages reported in the current GenSys session.

- Incremental Mode

  In this mode, GenSys report only those messages that were reported after this mode was turned on. To turn on the incremental mode, specify the following Tcl command:

  ```
  set_messaging_mode -incremental TRUE
  ```

You can select the type of mode from the View drop-down list, as shown below:



Waiving and Un-waiving Messages

If a particular message does not indicate a serious problem, you can waive that message so that it does not appear under the Messages tab.

GenSys displays all the waived messages under the Waiver Tree tab.

Waiving Messages

To waive a particular message, perform the following steps:

1. Right-click on the message.
2. Select the Waive option from the shortcut menu.

   A sub-menu appears.
3. Depending upon your requirement, select any of the following option from the sub-menu:
   - This Message: Select this option to waive only the selected message.
   - All Instances Of This Message: Select this option to waive all the similar messages.

After performing the above steps, all the waived messages appear under the Waiver Tree tab.

Un-Waiving Messages

If you want to unwaive a particular message or all the similar messages appearing under this tab, right-click on the message and select the Unwaive This Message or Unwaive All Instances Of This Message option, respectively, from the shortcut menu.

Storing Waived Messages of the Last Session

Details of waived messages of the last session are stored in the waivedMsg.tcl file, which is generated under the GenSysLogDir directory in the current working directory. GenSys reads this file when you invoke a new session and displays the waived messages of the last run under the Waiver Tree tab.

If you do not want the waived messages of the last session to be displayed under the Waiver Tree tab in the next session, delete the waivedMsg.tcl file before invoking the next session.

Note:
It is recommended to not modify this file.

Setting Log Filters

Log filters enable you to add filters on the processing messages appearing under the Log tab (by creating corresponding tabs in the Command/Log Window for each filter).

To set log filters, perform the following steps:

1. Right-click in the area under the Log tab.
2. Select the Log Filter option from the shortcut menu. The Log Filter dialog appears.
3. Specify the string for which the filter needs to be created. You may also specify regular expressions along with the filter string.

   A tab corresponding to the filter is created in the Command/Log Window.

For example, if you specify *add** as the filter string, all log or processing message corresponding to commands starting with add will be displayed under the add* tab.

Specifying Commands

The Command/Log Pane provides a *Command* text box in which you can specify Tcl commands to perform various actions. For example, if you want to create a component, specify the following Tcl command in the *Command* text box:

```
new component c2
```

All these commands are logged into a file, called gentclcmd.log, which is created inside the log directory present in the current working directory.

The Tcl log contains all Tcl commands executed. You can replay the commands in the Tcl log by using the Tcl Editor window. For an example of the Tcl log, refer to Tcl Log.

You can also specify the name of the Tcl file in the Command text box to execute all the Tcl commands present in that file.

Note:
For information on Tcl commands, refer to Tcl Commands Reference.

In addition, you can copy multiple Tcl commands from the *Command* text box, and paste them at the command prompt.

The *Command* text box has the following features:

- You can press the *Up* arrow key to display the last command entered.
- You can browse through the commands entered by using the *Up* and *Down* arrow keys.
- You can use the TAB key to complete a Tcl command in the *Command* text box.

Modular Schematic Window

The Modular Schematic window displays a graphical view (schematic layout) of the currently loaded design. It displays all instances and ports created in that design and connections between them.

By default, this window displays the schematic layout of the top-level design that shows components and other sub-designs present in the top-level design.

A sub-design of a design appears as a dotted box. Double-click on that sub-design to dive-into that sub-design. This opens the complete schematic of that sub-design to enable you view components instantiated in the sub-design.

Displaying the Modular Schematic Window

To display the Modular Schematic window, perform the following steps:

1. Load the required design/component under the Design Browser tab.
2. Select the Tools -> Modular Schematic menu option.

   Alternatively, select the Modular Schematic button (

) from the toolbar.

Navigating in the Schematic Using Click-Drag Combinations

Holding down the left mouse button and dragging the cursor causes changes in the views of Modular Schematic window.

When you release the mouse button, the view changes as per the direction in which the cursor is dragged.

The following table describes various views generate based on different directions in which the cursor is dragged:

Direction	Description
Northeast	Dragging the mouse with the left button pressed in this direction instructs the Modular Schematic window to decrease magnification of the schematic. The length of the dragging determines how far the Modular Schematic window will “zoom out.” A short drag in this direction will cause the viewing area to zoom out by a small multiplier, while dragging from one corner of the screen to the other will cause the viewing area to zoom out by a large multiplier. When the mouse button is released, the Modular Schematic window will zoom out with the selected modifier.
Southeast	Dragging the mouse with the left button pressed in this direction selects a “zoom in” area. When the mouse button is released, the Modular Schematic window will zoom until the area selected fills the window.
Southwest	Dragging the mouse with the left button pressed in this direction instructs the Modular Schematic window to increase or reduce magnification of the schematic as determined by the current settings. As a result, when the mouse button is released, the current module of the schematic will be completely visible in the Modular Schematic window. This is called “zoom to fit.”

Direction	Description
Northwest	Dragging the mouse with the left button pressed in this direction instructs the Modular Schematic window to “go up” the hierarchical tree and displays the parent design’s schematic. When the mouse button is released, the parent design’s schematic will be displayed with the zoom settings of the prior visit.

Using the Modular Schematic Window Menu Bar

The Modular Schematic window menu bar has the following options:

File	Edit	Tools

File

The File menu contains the following menu options:

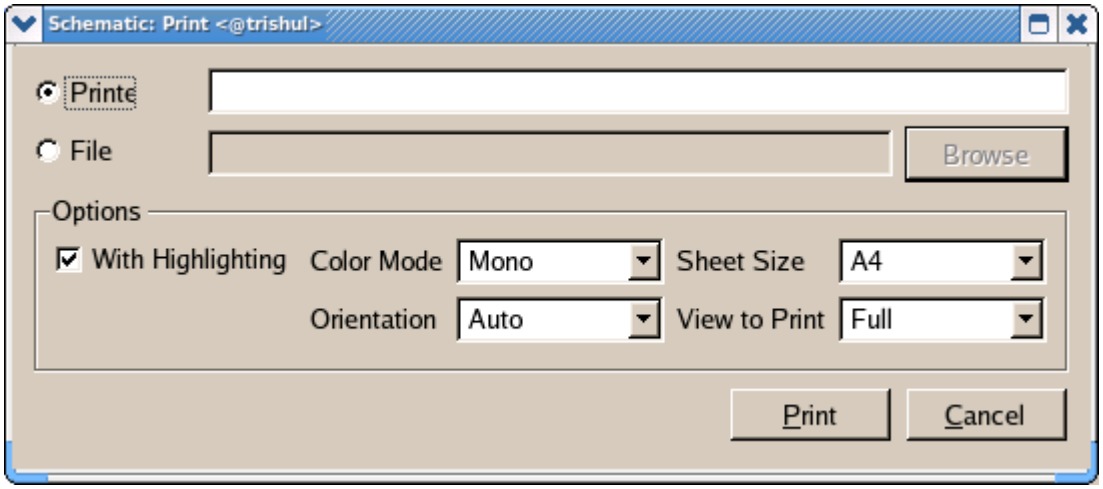
Print	Save Image	Exit

Print

Use this menu option to print the schematic or save it as a file.

When you select this menu option, the Schematic:Print dialog appears, as shown in the following figure:

Figure 11-19 Schematic:Print Dialog



The Schematic:Print dialog contains the following options:

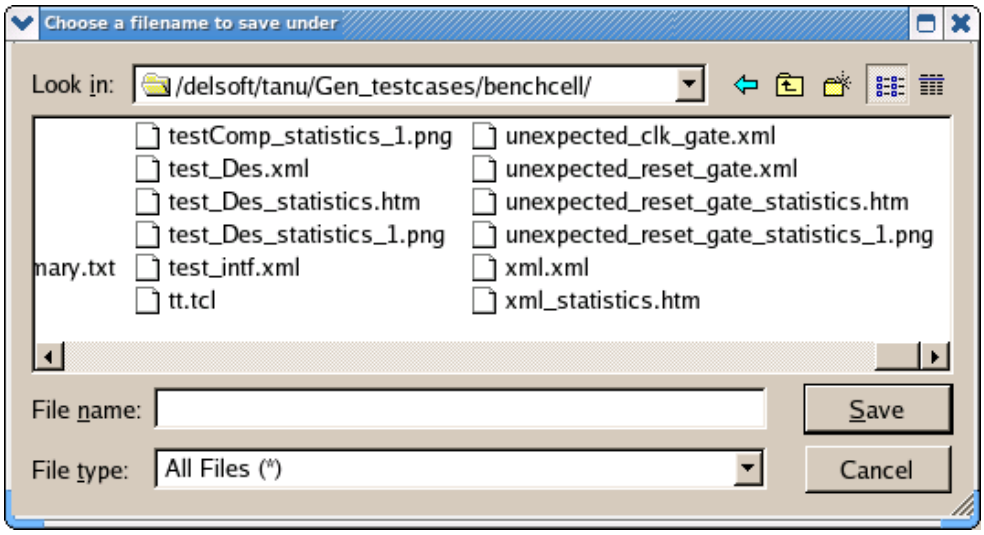
- Printer

  Select this option to print the schematic. Specify the print command specifying the path of the printer in the adjacent text box.

- File

  Select this option to save the schematic as a file. Specify the complete path name of the destination file in the adjacent text box.

  Alternatively, click the Browse button to navigate through the directory structure where you want to save the file. When you click this button, the Choose a filename to save under dialog appears, as shown in the following figure:



In the above dialog, browse to the required location, and append the directory path with a destination file name in the File name text box, and click the Save button.

Note:
Clicking Save in Choose a filename to save under dialog does not save the schematic to a file; it only sets the file name. You still need to click Print in the Print dialog to save the schematic to a file.

- Color Mode: Select Mono to print or save the schematic in monochrome mode, select Color to print or save the schematic in a color mode, or select Color(Inverted) to print or save the schematic in a reverse color mode.
- Orientation: Select Portrait to print or save the schematic in a portrait orientation, select Landscape to print or save the schematic in a landscape orientation, or select Auto to let the software decide the best-fit orientation.
- Sheet Size: Select the target paper size from the supported paper sizes.
- View to Print: Select Full to print or save the full schematic or select Current to print or save the currently displayed view of the schematic.
- With Highlighting: If you have selected to print or save the schematic in Color mode, you can set this option to print the highlighted objects with their respective colors.

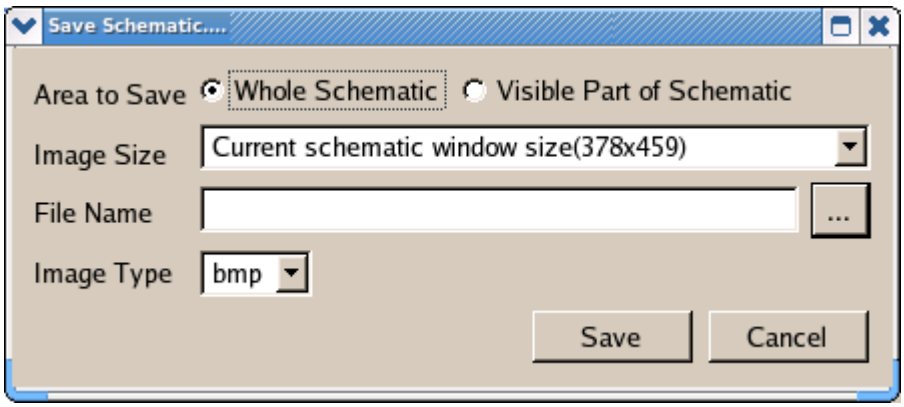
  Note:
  The With Highlighting option will be functional in future release of GenSys.

Save Image

Use this menu option to save the schematic as an image.

This displays the Save Schematic dialog, as shown in the following figure:

Figure 11-20 Save Schematic Dialog



You can specify the following settings in this dialog:

- Specify the area of the schematic that needs to be saved as an image.

If you select the Whole Schematic option, GenSys saves the complete canvas view of the entire design as an image. However, if you select the Visible Part of Schematic option, GenSys saves only the visible portion of the schematic as an image. For example, if you want to save the image of one of the instance appearing the schematic, zoom that instance in the Modular schematic window so that only that instance is visible in the available space of the window. Then, select the Visible Part of Schematic option in the Save Schematic dialog.

- Specify the size of the image.

  You can select the desired size of the image from the Image Size drop-down list.

- Specify the image type

  You can specify the image type (bmp, png, or jpeg) by selecting the required option from the Image Type drop-down list.

- Specify the name of the image

You can specify the complete path and name of the image in the File Name text box. Alternatively, click the Browse button to browse to the directory where you want to save the image. When you click the Browse button, GenSys displays the Choose a filename to save under dialog in which you can select the directory where the image is to be saved, and specify the name of the image.

Exit

Use this menu option to close the Modular Schematic window.

Edit

The Edit menu contains the Find menu option. For details, see Searching Objects in the Schematic.

Tools

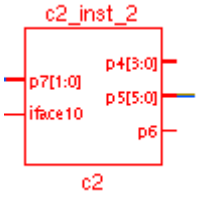
The Tools menu contains the following menu options:

Elaborate	Planes	Hide/Show Toolbar Icons	Options

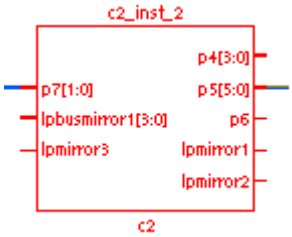
Elaborate

Use this menu option to view elaborated modular schematic of the design. When you use this option, the adhoc connections between the pins of the interfaces and pins of other instances are created and shown in addition to other connections created by you.

For example, by default, a component with an interface instance is displayed in the modular schematic as shown below:



After elaboration of design, all the connections with the pins of interface are also displayed as shown below:



Note:

This option can be used only when the Show Interface Ports option is selected in Incremental Schematic Window. However, if you have generated RTL before using this menu option, it is not essential to select the Show Interface Ports option. In this case, the Modular Schematic Window takes data about adhoc connections from the generated RTL.

Planes

Use this menu option to open the Planes dialog. For details, see .

Hide/Show Toolbar Icons

Use this menu option to display the Hide/Show Icons dialog. Use this dialog to hide/show buttons appearing in the schematic toolbar.

Options

Use this menu option to open the Schematic:Options dialog. Use this dialog to specify various color settings and certain miscellaneous options in the schematic.

For details, see Updating Color Settings and Miscellaneous Settings.

Incremental Schematic Window

The *Incremental Schematic* window displays the selected portions of the (flattened) design schematic across hierarchical boundaries.

To open this window, perform any of the following actions:

- Select an instance in the Canvas View, and click the *Incremental Schematic* button (

-) on the toolbar.
- Load an object in the *Incremental Schematic*. See Load to IS.
- Add an object in the *Incremental Schematic*. See Add to IS.

Adding Objects in Incremental Schematic

You can add objects in Incremental Schematic in any of the following ways:

- Loading the selected objects after clearing the previous view of Incremental Schematic

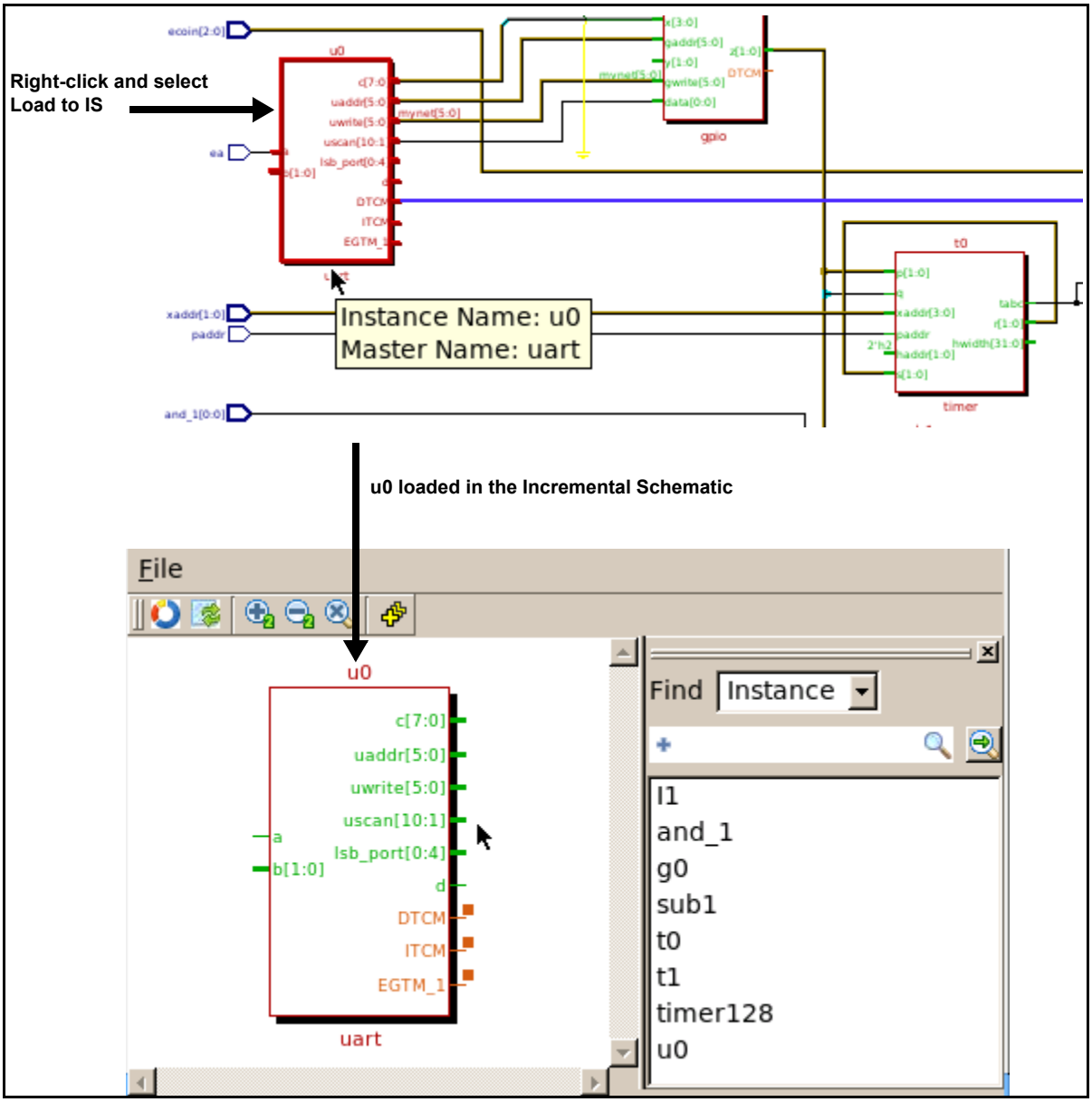
  See Load to IS.
- Loading the selected objects incrementally in the Incremental Schematic

  See Add to IS.

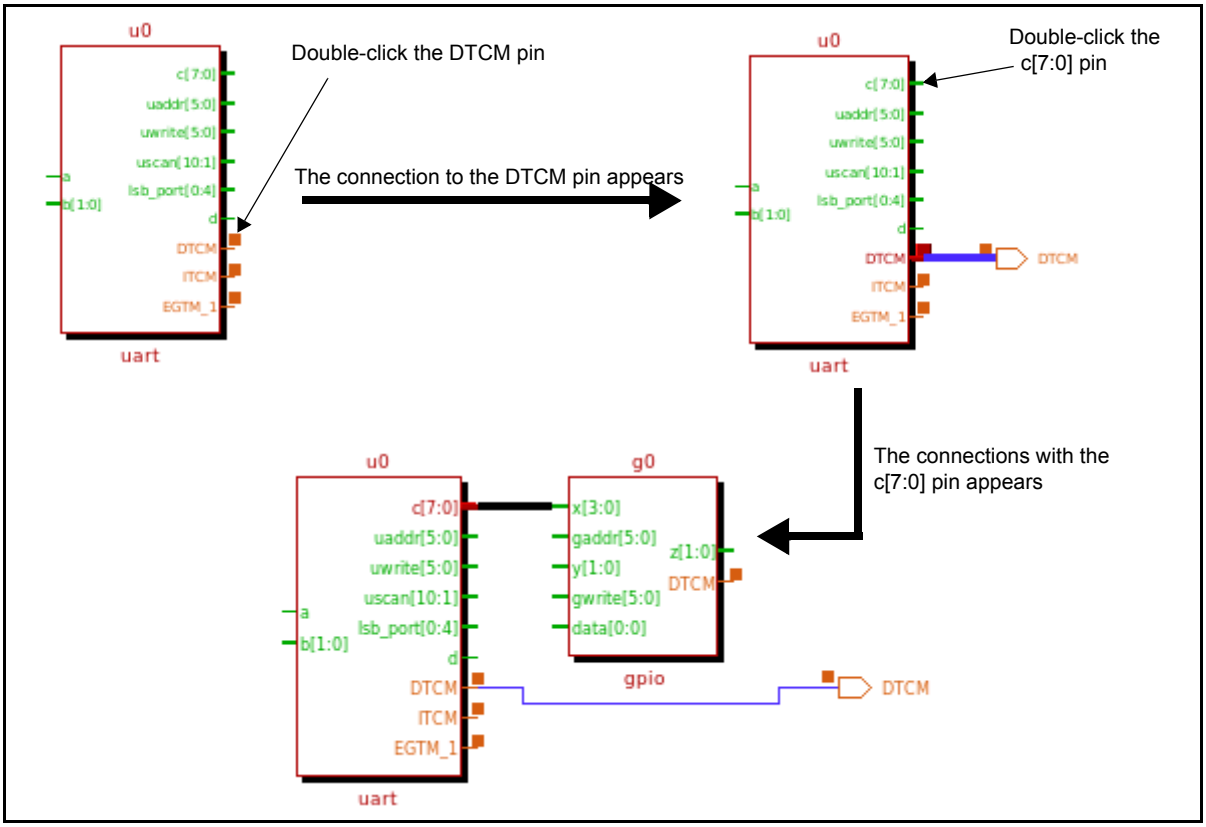
Load to IS

To load an object after clearing the previous view of Incremental Schematic, right-click on the object in the *Canvas View* or under the *Design Browser* tab, and select the *Load to IS* option from the shortcut menu.

The following figure shows the example of loading an object from the *Canvas View* to the *Incremental Schematic*:



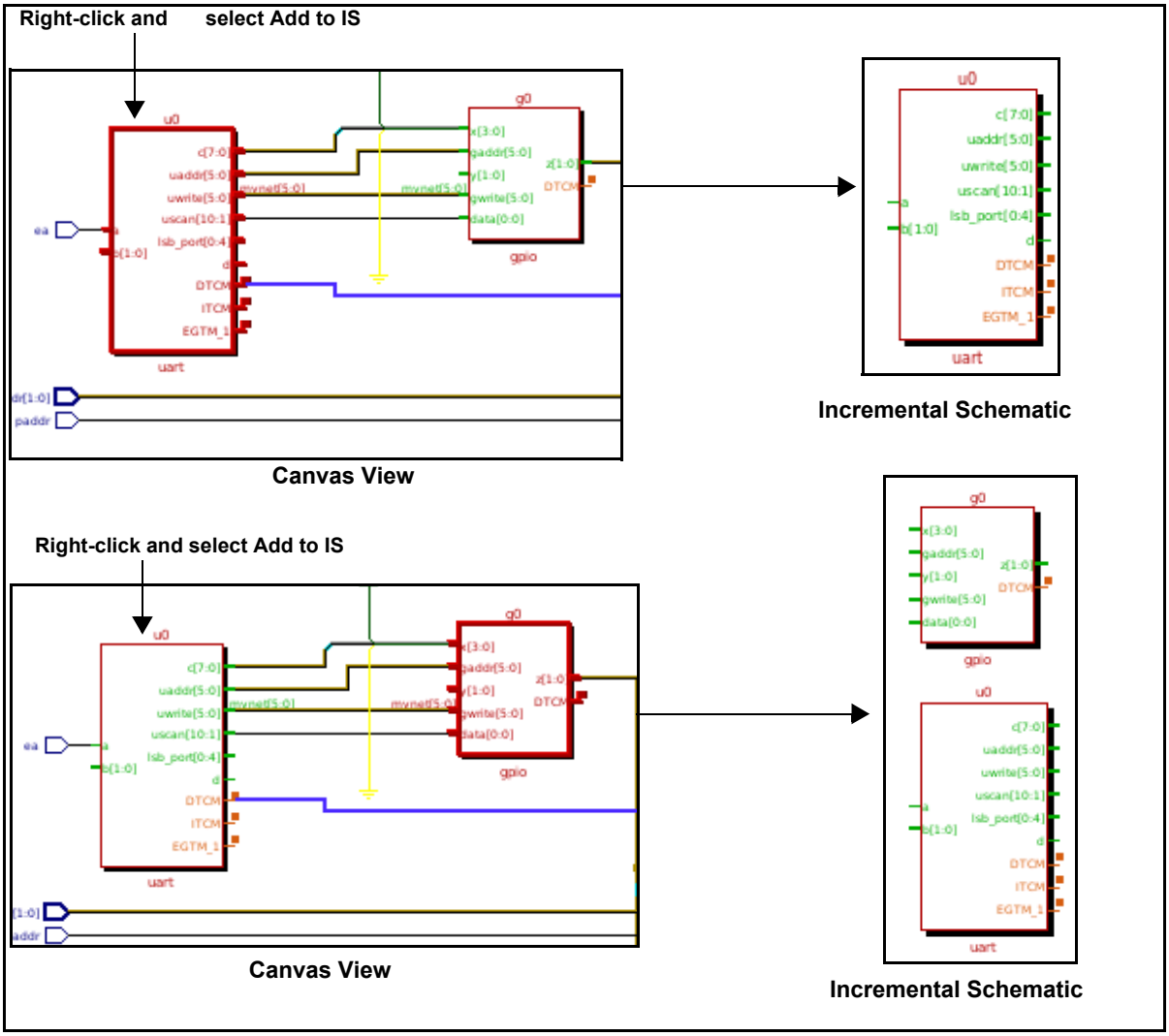
After loading the object, as shown in the figure above, you can trace the connectivity of the object by double-clicking the pins. For details, see the following figure:



Add to IS

To incrementally add objects in the schematic, right-click on an object in the *Canvas View* or under the *Design Browser* tab and select the *Add to IS* option from the shortcut menu.

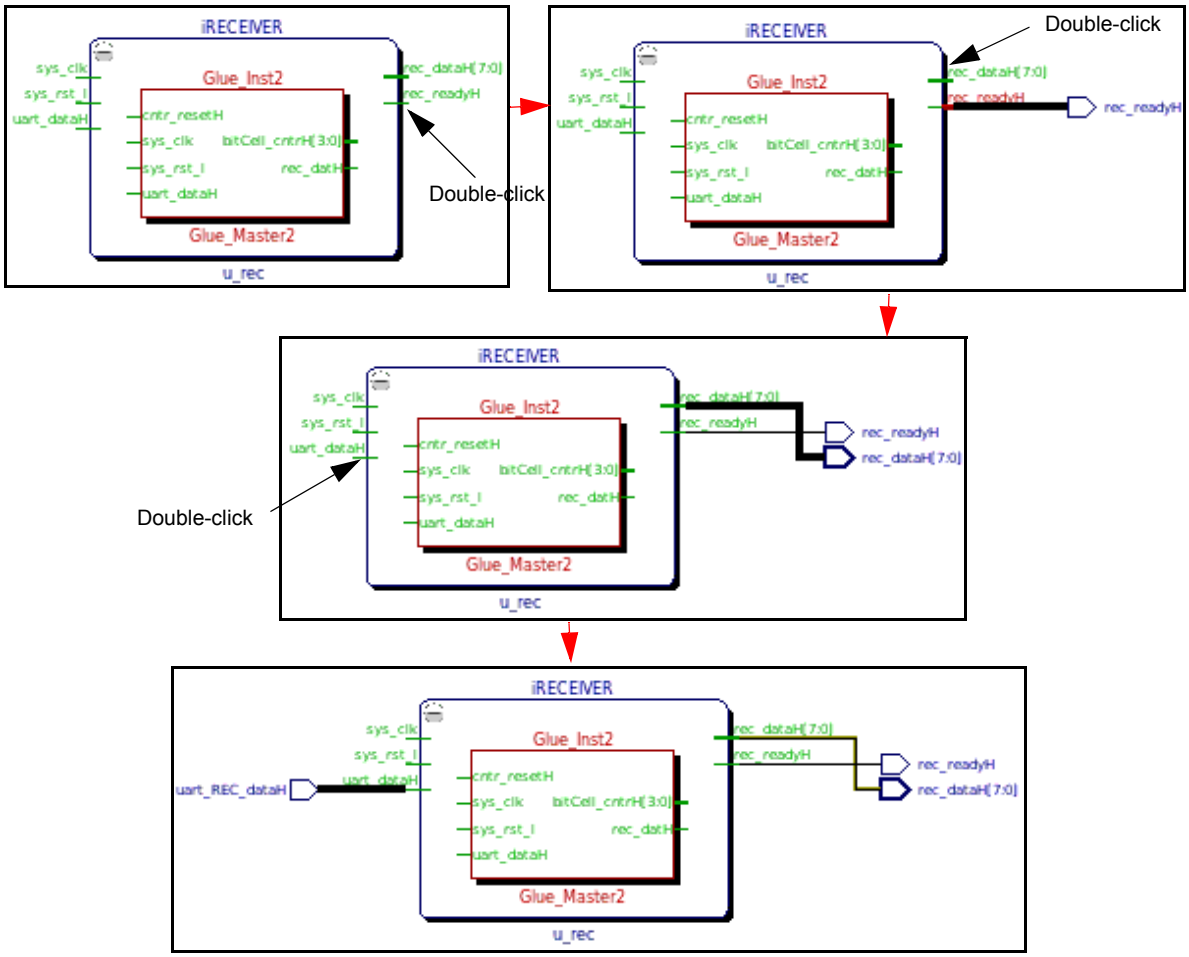
The following figure shows the example of adding objects incrementally in the *Incremental Schematic*:



Viewing Connections in Incremental Schematic Window

After adding instances in the schematic area, double-click on the pin/interface of an instance to view connections of that instance with other instances/ports through that pin/interface.

This is shown in the following figure:



Navigating through the Incremental Schematic Window

You can navigate through the Incremental Schematic window by using the same techniques as used in Modular Schematic window. (See Navigating in the Schematic Using Click-Drag Combinations).

Specifying Incremental Schematic Options

You can modify the color settings and certain miscellaneous options in the schematic.

To update the schematic options, right-click anywhere in the Incremental Schematic and select *Options* from the shortcut menu. This displays the *Schematic:Options* tab of the

GenSys:Options dialog. Under this tab, you can specify the required settings. For details on this dialog, see Options.

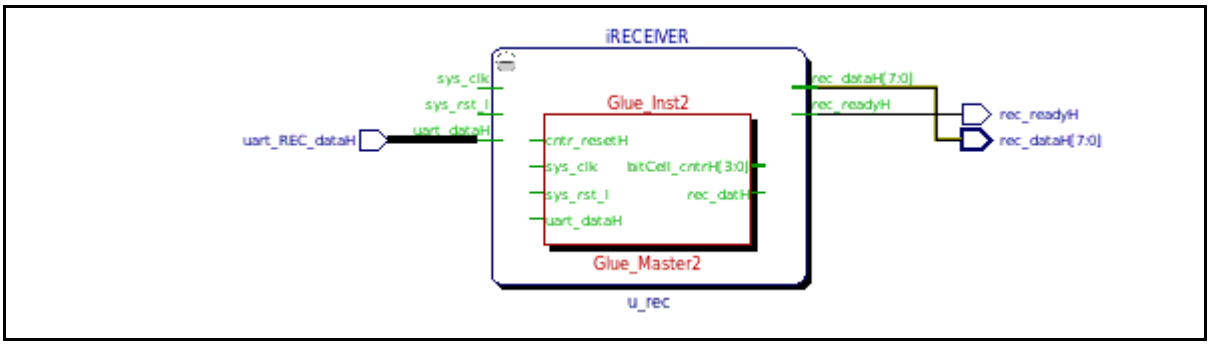
Right-Click Menu Options on the Blank Area of Incremental Schematic

When you right-click on a blank area of Incremental Schematic, the following options appear in the shortcut menu:

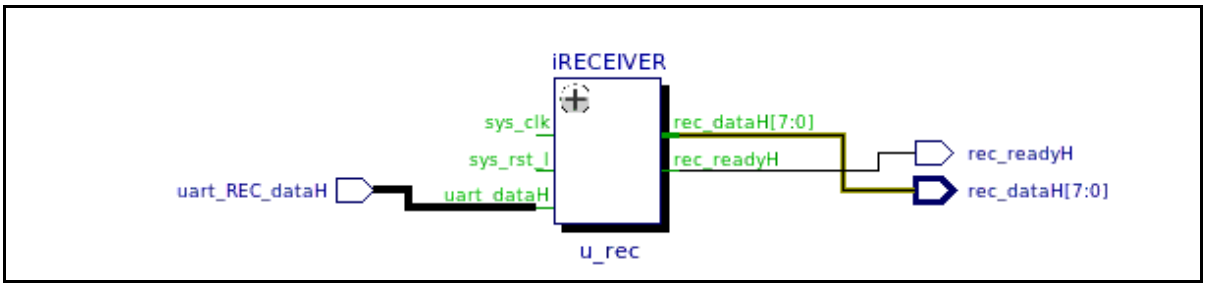
- *Redraw Nets*: Redraws entire connectivity in the best optimized way.
- *Clear*: Clears the Incremental Schematic window
- *Collapse All Instances*: Collapses all the instances that have a hierarchy displayed inside them.

  For example, consider the following Incremental Schematic containing an instance that displays a hierarchy within itself:

\



In the above schematic, when you right-click on a blank space and select the *Collapse All Instances* option, you see the following schematic:



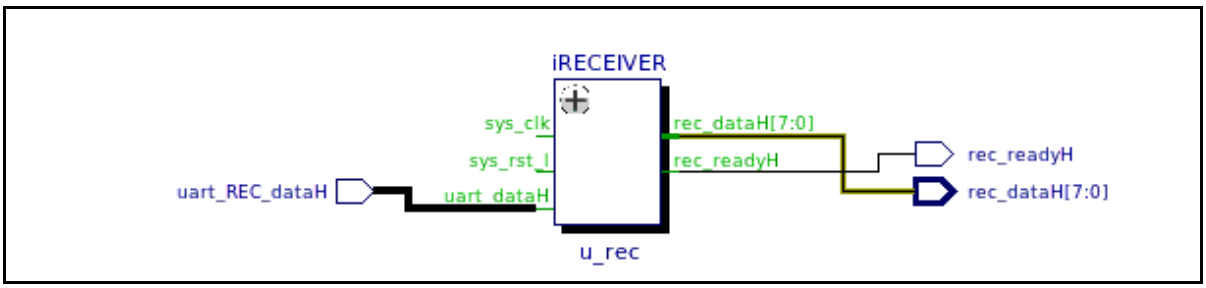
To bring back the above schematic in the expanded form, again right-click on the blank space and now select *Expand All Instances*.

- *Hide Unconnected Pins From All Instances*: is a global option which will hide all unconnected pins from the instances
- *Options*: Opens the *Schematic:Options* tab of the *GenSys:Options* dialog. Under this tab, you can specify the required schematic settings. For details on this tab, see Options.

Right-Click Menu Options on an Instance in Incremental Schematic

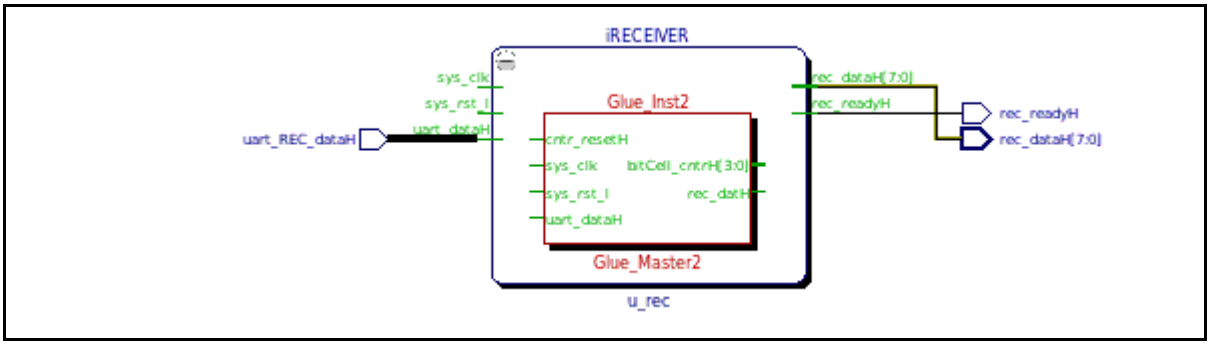
When you right-click on an instance in the Incremental Schematic, the following options appear in the shortcut menu:

- *Clear Except This*: Clears all the other instances from the schematic except the selected instance on which you right-clicked the mouse button.
- *Hide Unconnected Pins*: Hides the unconnected pins of the selected instance.
- *Redraw Nets*: Redraws entire connectivity in the best optimized way.
- *Clear*: Clears the Incremental Schematic window
- *Expand*: Expands the selected instance to show the instance hierarchy. For example, consider the following instance:



When you right-click the above instance and select the *Expand* option, it changes to the following:

\



To collapse the above instance to hide its hierarchy, right-click on the instance and select the *Collapse* option.

- *Options*: Opens the *Schematic:Options* tab of the *GenSys:Options* dialog. Under this tab, you can specify the required schematic settings. For details on this tab, see Options.

Right-Click Menu Options on a Pin in Incremental Schematic

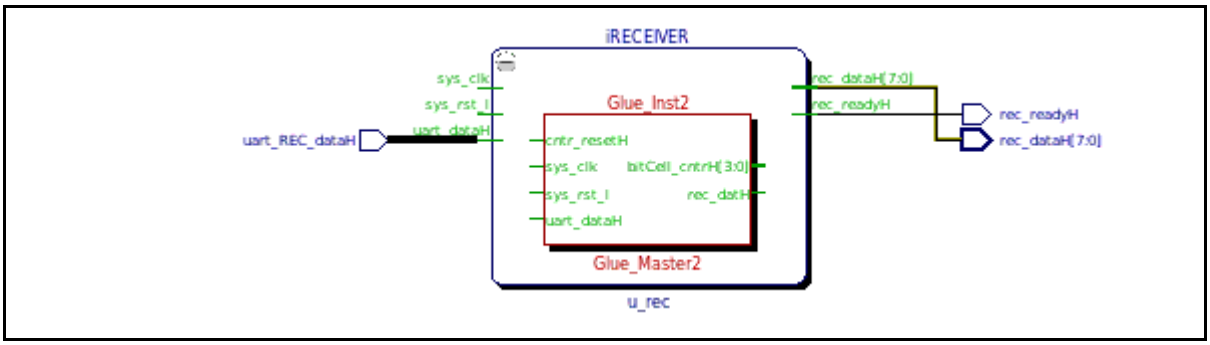
When you right-click on a pin in the Incremental Schematic, the following options appear in the shortcut menu:

- *Show Input Cone*: Shows the connectivity from the selected input pin till any of the following (whichever is applicable)
  - The top-level design port
  - Last connectivity is found on the input side
  - A leaf-level component
- *Show Output Cone*: Shows the connectivity from the selected output pin till any of the following (whichever is applicable)
  - The top-level design port
  - Last connectivity is found on the input side
  - A leaf-level component
- *Clear Except This*: Clears all the other instances from the schematic except the instance of the selected pin.
- *Clear*: Clears the Incremental Schematic window

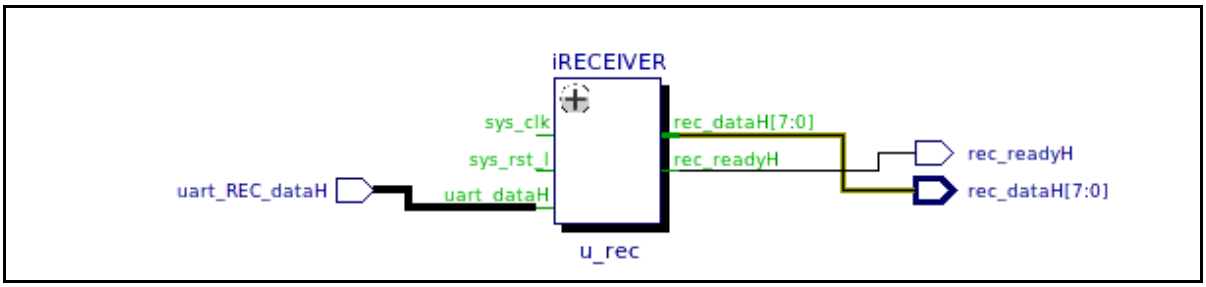
- *Collapse All Instances*: Collapses all the instances that have a hierarchy displayed inside them.

  For example, consider the following Incremental Schematic containing an instance that displays a hierarchy within itself:

\



In the above schematic, when you right-click on a blank space and select the *Collapse All Instances* option, you see the following schematic:



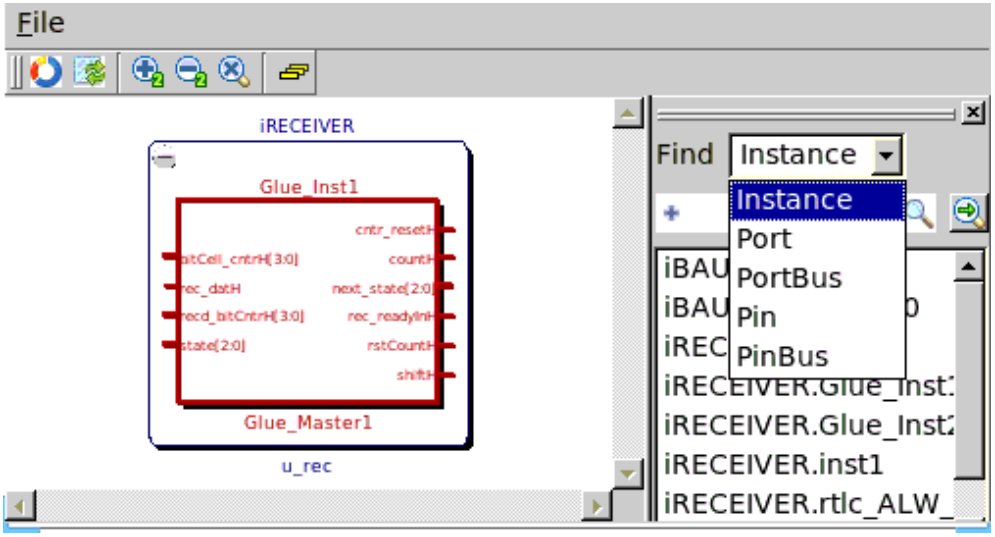
To bring back the above schematic in the expanded form, again right-click on the blank space and now select *Expand All Instances*.

- *Hide Unconnected Pins From All Instance*: Hides the unconnected pins of all instances.
- *Options*: Opens the *Schematic:Options* tab of the *GenSys:Options* dialog. Under this tab, you can specify the required schematic settings. For details on this tab, see Options.

Finding Objects in Incremental Schematic

Use the *Find* section of the Incremental Schematic window to find different types of objects, such as instances, pins, pin bus, port, and port bus.

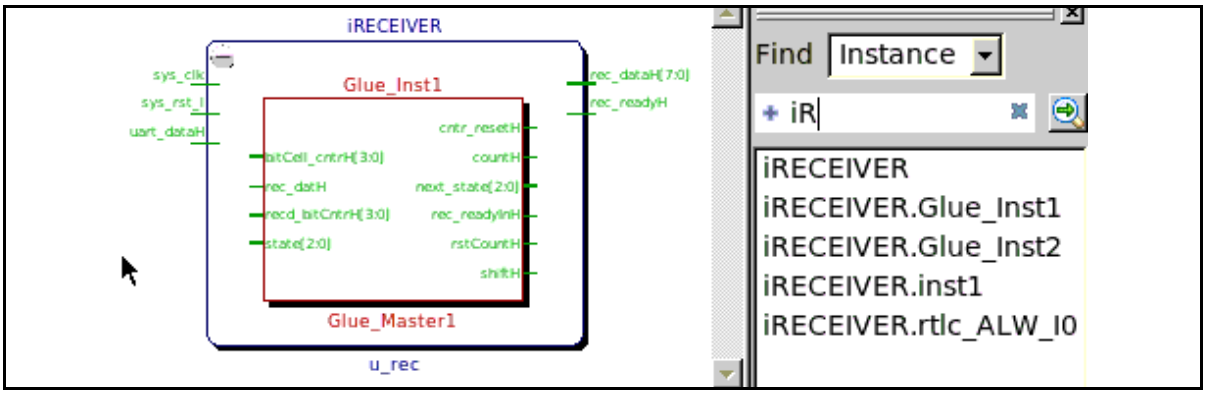
The following figure shows the *Find* section of Incremental Schematic:



In the *Find* section in the above figure, select the object type to be searched from the drop-down list and then specify the search string in the textbox below the drop-down list.

Consider that you want to find an instance starting with the string iR in the above schematic. Perform the following steps to find this object:

1. Select Instance from the drop-down list.
2. Type iR in the textbox below the drop-down list, as shown in the following figure:

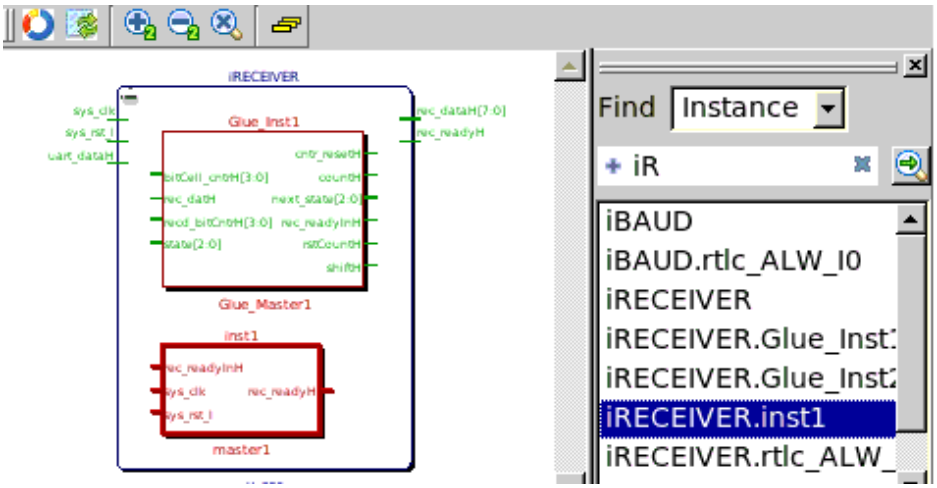


Note that as you type iR, the filtered list instances starting with this string appear in the list below.

3. Select iRECEIVER.inst1 from the filtered list of instances.
4. Click the

5. icon to locate that instance in the schematic.

After performing the above steps, the iRECEIVER.inst1 instance appears in the schematic, as shown in the following figure:



Statistics Window

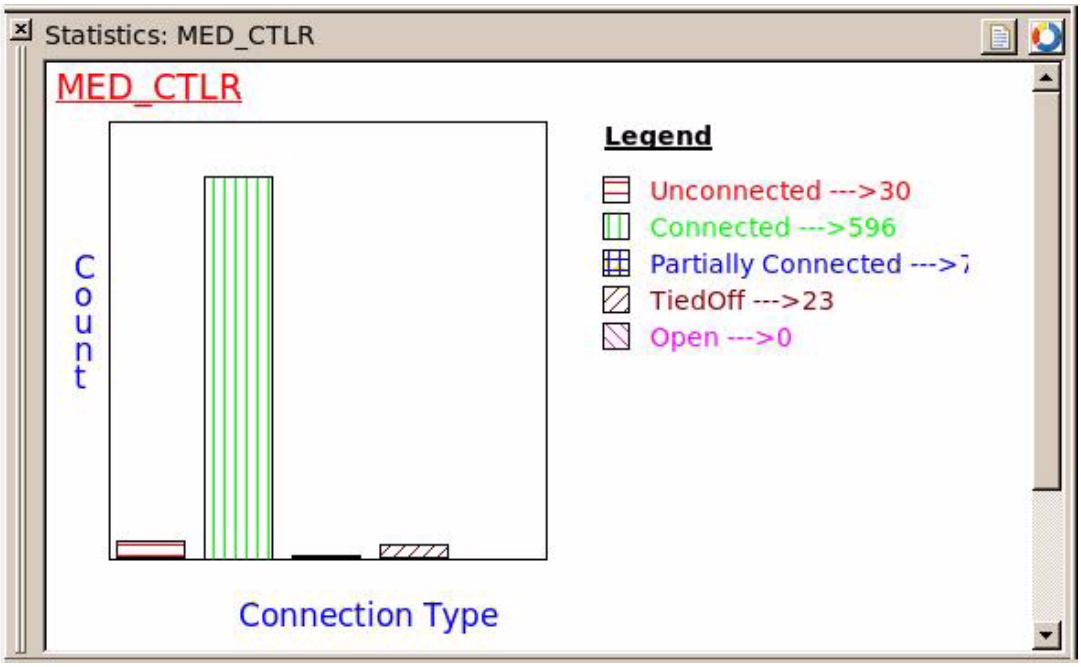
This window appears at the bottom of the main GenSys window, and displays the following information:

- Number of connections made
- Number of unconnected ports/terminals
- Number of partially connected
- Number of tie-off/open connections
- Number of Subsystems
- Number of leaf level instances

Statistics window appears in the Command/Log Pane towards the left side. You can un-dock this window and place it anywhere on the desktop.

You can view this report by selecting the View -> Views -> Statistics Window menu option.

The following figure shows the statistics window:



You can modify connections in a design while keeping the Design Statistics report open.

The above report may not show updated results in some cases. To view the updated report, click the

icon in the report. GenSys first elaborates the design and then creates the updated report. This is very useful as the design assembly is done step by step.

Note:

GenSys also generates an HTML file for the Statistics window that you can open from any internet browser.

Menu Bar

GenSys menu bar is categorized in the following menus:

File Menu	Edit Menu	View Menu	Library Browser Menu
Generator Menu	Tools Menu	Reports Menu	Help Menu

File Menu

Following options appear when you click the File menu:

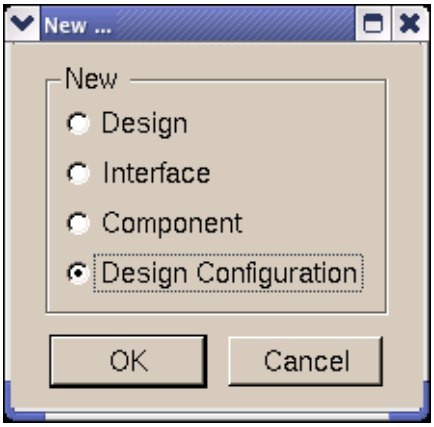
File > New	File > Open	File > Save
File > Export	File> Save All	File > Export All
File > Save Session	File > Restore Session	File > Close
File > Close All	File > Print Schematic	File > Load RTL
File > Set Working Directory	File > Set Output Directory	File > Set XML Directory
File > Set Save Template	File > Set Reports Directory	File > Set Netlist Directory
File > Exit		

File > New

This menu option is used to create a new design, interface, component, or design configuration.

When you select this menu option, the New dialog appears, as shown in the following figure:

Figure 11-21 New Dialog



In the above dialog, select the required option (Design, Component, Interface, or Design Configuration), and click the OK button. When you click the OK button, the New File dialog appears in which you can specify the name of .xml file of the design, component, interface, or design configuration to be created.

File > Open

This menu option is used to open a design, component, interface, or design configuration in the IP-XACT, MRS, or Tcl format, and load it in the Graphical User Interface (GUI).

When you select this menu option, the Select one or more files to open dialog appears. In this dialog, you can navigate through the directory structure to locate the required file. You can select the type of the file that you need to open by selecting the required type from the File type drop-down list. Once you locate the required file, click the Open button to open that file.

Note:

If two components/designs/interface/design configuration with the same VLNV are lying on same path, then the one with latest time stamp is picked to be loaded in the GUI.

The keyboard shortcut to open a design is CTRL+O.

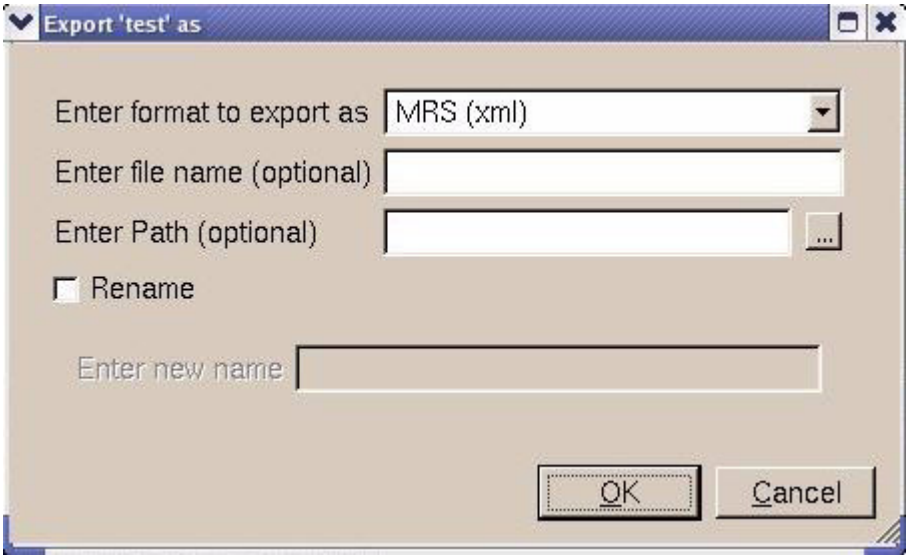
File > Save

This menu option is used to save the design, component, interface, or design configuration.

File > Export

This menu option is used to save the currently loaded design, component, interface, or design configuration with a different name and format. when you select this menu option, the following dialog appears:

Figure 11-22 Save As Dialog



In the above dialog, select the required format in the Enter format to save as drop-down list in which you want to save the file. You can select any of the following formats from this drop-down list:

MRS (xml)	IPXACT 1.4 (xml)	IPXACT 1.2 (xml)	Tcl (cmd/tcl)

You can also specify a file name for the design, component, interface, or design configuration being saved in the Enter file name (optional) text box.

In addition, you can also change the name of design, component, interface, or design configuration by checking the Rename check box. This enables the Enter new name text box in which you can specify the new name for the selected design, component, interface, or design configuration.

Note:

Whenever you change the VLNV of an object and save it, GenSys displays the Confirm Overwrite dialog to confirm if you want that object to be overwritten with the new VLNV information.

File> Save All

This menu option is used to save all the items that are currently loaded in the GUI in the directory that is set as the current working directory.

File > Export All

This menu option is used to save all the xml files in the selected destination directory.

When you select this menu option, the Choose Destination Directory dialog appears. In this dialog, select a directory in which you want to save the files. You can also select the file type from the File type drop-down list in which you want to save the files. GenSys allows you to save the files in the following types:

MRS	AutoRTL	IPXACT	Tcl

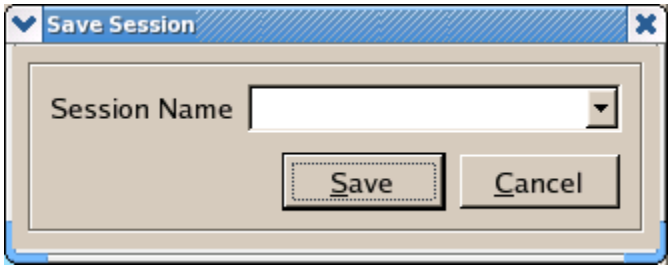
Click the OK button to save the files.

Note:
If you do not use this menu option, GenSys saves files in the db directory.

File > Save Session

This menu option is used to save the current session.

When you select this menu option, the following dialog appears:



In the above dialog, specify a name for the current session to be saved and click the Save button.

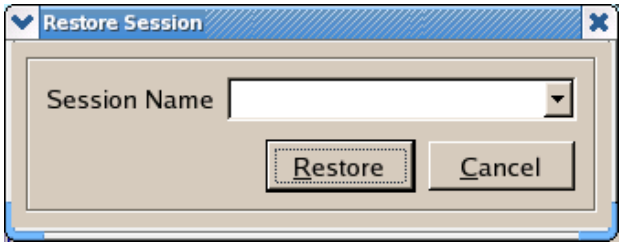
When you click this button, GenSys takes a snapshot of the entire database and saves it in a hierarchical NVLV database format. This database is saved under the <current-working-directory>/.gensys_dir/gensys_states/<session-name> directory.

You can later restore the saved session by clicking the File > Restore Session menu option.

File > Restore Session

This menu option is used to restore a session previously saved by using the File > Save Session menu option.

When you select this menu option, the following dialog appears:



In the above dialog, specify the name of the session to be restored and click the Restore button.

File > Close

This menu option is used to close the currently loaded design, component, interface, or design configuration in the GUI. In addition, all the components, designs and interfaces that were loaded automatically by that design, component, or interface are also closed.

File > Close All

This menu option is used to close all the currently loaded designs, components, interfaces, or design configurations in the GUI. It also removes all the currently loaded objects from the memory.

When you select this menu option, all the internal pointers earlier returned by Perl APIs/OO methods or Tcl APIs become inaccessible. Hence, you should not use such (stored) pointers after selecting this menu option.

File > Print Schematic

This menu option is used to print the schematic or save it as a file. For details, see Print.

Note:
This menu option appears only in the Canvas View.

File > Load RTL

This menu option is used to open the Select RTL/InterfaceLib Files dialog that is used to import RTL.

For details on importing RTL, see Performing RTL Import.

File > Set Working Directory

This menu option is used to set the working directory. The files generated by the generators are created in this directory. (The default for CWD is ‘./’).

When you select this option, the Choose Working Directory window appears. In this window, select the desired directory and click the OK button.

File > Set Output Directory

This menu option is used to set the output directory. All new XML files and internal generator output files are created in the output directory. By default, GenSys considers the working directory as the output directory.

When you select this option, the Choose Output Directory window appears. In this window, select the desired output directory and click the OK button.

File > Set XML Directory

This menu option is used to set the XML directory. All the new XML files are created in the XML directory.

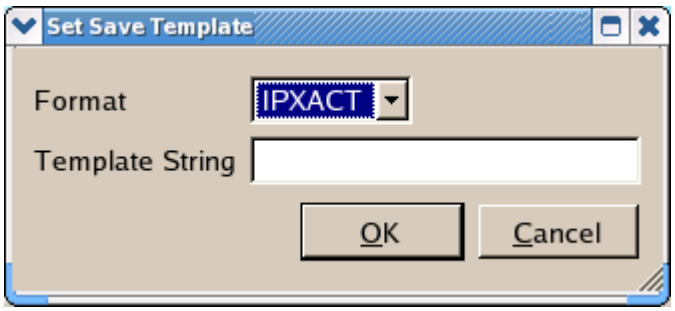
Note:
If you do not use this menu option, GenSys saves files in the db directory.

When you select this option, the Choose XML dump Directory window appears. In this window, select the desired XML directory and click the OK button.

File > Set Save Template

This menu option is used to set a directory structure for a specific file type.

When you select this option, the Set Save Template dialog appears, as shown in the following figure:



In the above dialog:

- Select a file format, such as IPXACT, MRS, or TCL from the Format drop-down list.
- Specify a string in the Template String text box to specify a directory structure in which a file of the selected format should be saved.

  ```
  "[<relative-dir>]
  [%vendor<separator>][%library<separator>][%name<separator>][%version<s
  eparator>]%obj.<ext>"
  ```

  Note:

  If you do not use this menu option, GenSys saves files in the db directory.

File > Set Reports Directory

This menu option is used to set the reports directory to store the generated reports from GenSys.

When you select this option, the Choose Reports Directory window appears. In this window, select the desired directory and click the OK button.

File > Set Netlist Directory

This menu option is used to set the netlist directory to store the Verilog/VHDL files generated after running the Verilog/VHDL generator.

When you select this option, the Choose Netlist Directory window appears. In this window, select the desired directory and click the OK button.

You can set a netlist directory using the set_netlist_dir Tcl command also. If the netlist directory is not present, Verilog and VHDL files are saved in the output/VHDL and output/Verilog directories, provided that you have not specified directories by using the set_vhdl_dir and set_verilog_dir Tcl commands.

File > Exit

This menu option is used to close the GenSys application.

When you select this option, the Save window appears prompting you to save the changes made in the GenSys application. Click the Yes button to save the changes.

Edit Menu

Based on the current view, tabular or schematic, appropriate menu options appear in the Edit menu.

Menu Options in the Tabular View

The following menu options appear in the Edit menu when the current view is the tabular view:

Cut	Copy	Paste
Clear Contents	Select All	Fill with Defaults
Insert Row	Insert SubTable	Delete Row
Set Table Defaults	Configure	Find

Cut

This menu option is used to cut the currently selected rows in a table. The keyboard shortcut for this menu option is CTRL+X.

Copy

This menu option is used to copy the currently selected rows in a table. The keyboard shortcut for this menu option is CTRL+C.

Paste

This menu option is used to paste the already cut or copied rows in other tables. The keyboard shortcut for this menu option is CTRL+V.

Clear Contents

This menu option is used to clear the contents of a table in the Data Table window. The keyboard shortcut for this menu option is CTRL+SHIFT+c.

Select All

This menu option is used to select all the rows in the Data Table window. The keyboard shortcut for this menu option is CTRL+A.

Fill with Defaults

This menu option is used to fill the selected cells with the default values set for that table. You can specify the default values by using the Set Table Default option.

Insert Row

This menu option is used to insert a new row with default values after the currently selected row in a table. The keyboard shortcut for this menu option is the Insert key.

Insert SubTable

This menu option is used to insert a subtable in the selected row (for example, bitfields for a register, bitEnum Values for a given bitfield). The keyboard shortcut for this menu option is CTRL+T.

Delete Row

This menu option is used to delete all the selected rows from the table in focus.

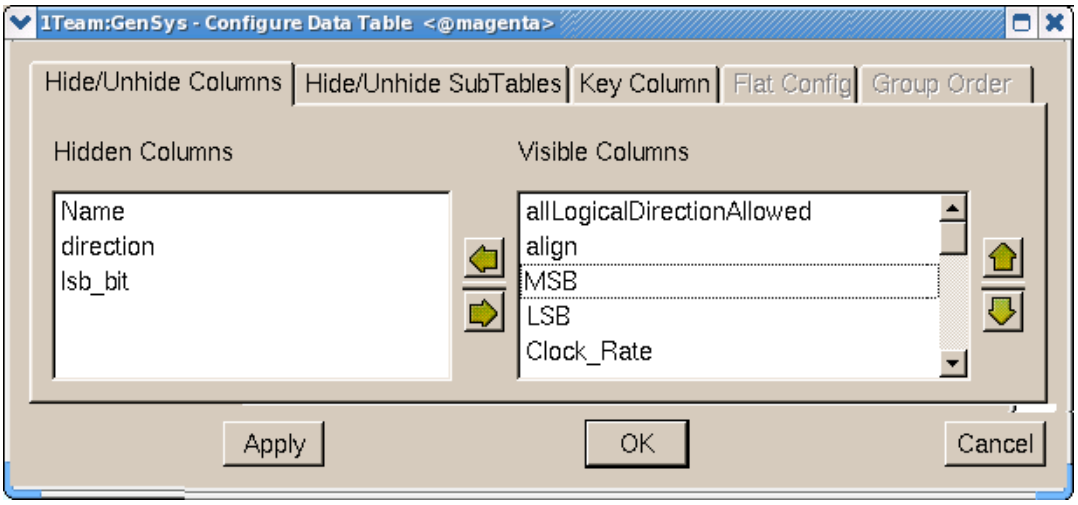
Set Table Defaults

This menu option is used to fill the default values for the focussed table. These values are used to fill values in the newly inserted rows. Cells which are already filled are not affected.

Configure

This menu option is used to hide/unhide columns and sub tables, sort columns, and specify the key columns. When you select this option, the Configure Data Table dialog appears, as shown in the following figure:

Figure 11-23 Configure Data Table



Note:

The attributes and dataSources subtables of all the tables are hidden by default. You can unhide them using this menu option.

Find

This menu option is used to search for items that matches with the specified text.

Search will not try to move inside the nested tables. To search a text inside a nested table, you have to invoke the Find dialog with an item selected (i.e. that table needs to be in focus) in that nested table. The keyboard shortcut for this menu option is CTRL+F.

Menu Options in the Canvas View

Note:

Menu options in the Edit view are enabled only when the canvas view is in the edit mode.

The following menu options appear in the Edit menu when the current view is the canvas view:

Add Port	Add Connection	Delete Object	Export Pin/ Interface
Copy Object Name			

Add Port

Use this menu option to add a port on the objected selected in the schematic.

When you click this menu option, the Add Port dialog appears in which you can specify the required port information. For details, see Adding Ports.

Add Connection

Use this menu option to add a connection between the selected pins/ports in the schematic.

When you click this menu option, the Connections Options dialog appears in which you can specify the required connection information. For details, see Creating Connections with Additional Details.

Delete Object

Use this menu option to delete the selected object.

Export Pin/Interface

Use this menu option to export selected pins/ports to a design.

When you click this menu option after selecting a pin/port, the Export Pin dialog appear in which you can specify the required information. For details, see Adding Glue Logic in a Design.

Copy Object Name

Use this menu option to copy the name of the selected object.

View Menu

When you click the View menu, GenSys displays various options, as shown in the following figure:

Goto Next Error	Goto Previous Error	Views

Goto Next Error

This menu option is used to go to the next error shown after validation of the tables.

Note:
This menu option appears only in the tabular view.

Goto Previous Error

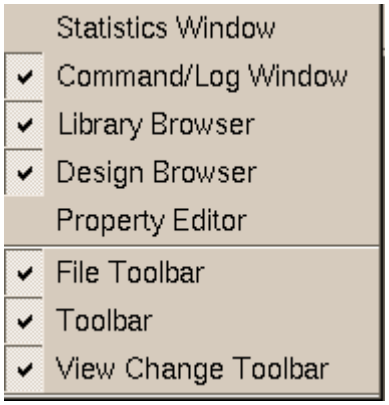
This menu option is used to go to the previous error shown after validation of the tables.

Note:
This menu option appears only in the tabular view.

Views

This menu option displays a sub-menu, as shown in the following figure:

Figure 11-24 View Sub-Menu



In the above menu, you can select/deselect various windows/menus that you want to show/hide in GUI.

Note:
The same menu also appears when you right-click on the toolbar.

Library Browser Menu

When you click the Library Browser menu, GenSys displays the following options:

Add	Add/Remove Files With VLNV	Remove From Library
Edit Search Path	Scan Directory	

Add

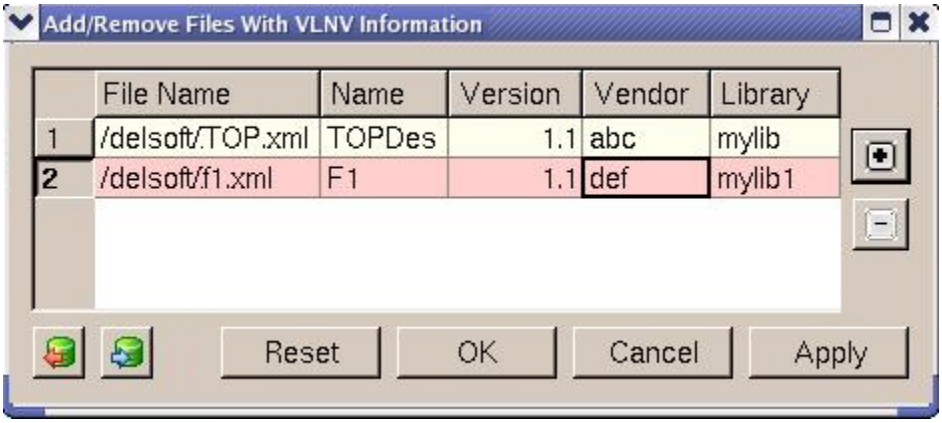
This menu option is used to add information of the .xml file to the Library Browser. This menu option is used to add a single .xml file to the library browser.

Add/Remove Files With VLNV

This menu option is used to add or remove files with their VLNV information from the Library Browser. In addition, this option is also used to load and dump the VLNV database file.

When you select this option, the Add/Remove Files With VLNV Information dialog appears, as shown in the following figure:

Figure 11-25 Add/Remove Files With VLNV Information Dialog



This dialog displays all the those lib files for which the VLNV information is present. Here, you can add/modify/delete the file paths and the VLNV information for each file.

This dialog contains various buttons using which you can perform different functions, as specified in the following table:

Button	Function
	This button is used to add files along with their VLNV information in the Library Browser. When you click this button, GenSys adds a new row in this dialog. To specify a file in this newly added row, click the File Name field in that new row. This displays the button adjacent to the File Name field. Click this button to open the Add File dialog, in which you can select the file to be added in the Library Browser. You can then specify the VLNV information for this file in the appropriate fields.
	This button is used to remove the selected files from the Library Browser.
	This button is used to load the VLNV database file (XML format). When you click this button, GenSys displays the Choose File dialog from where you can select the VLNV database file (in XML format) to be loaded. When you load a VLNV database file, GenSys reads all the files along with their VLNV information present in that database file, and adds all those files in the Library Browser.
	This button is used to specify the VLNV database file in which GenSys dumps all the files present in the Library Browser along with their VLNV information in the XMl format. When you click this button, the Choose File dialog appears from where you can select the VLNV database file. If you want to save the relative file paths in the VLNV database file, select the Keep relative paths check box in the Choose File dialog.

Once you specify the required settings by using the above buttons, click the OK or Apply button to save all those changes. To loose all the modifications made in this dialog, click the Reset button.

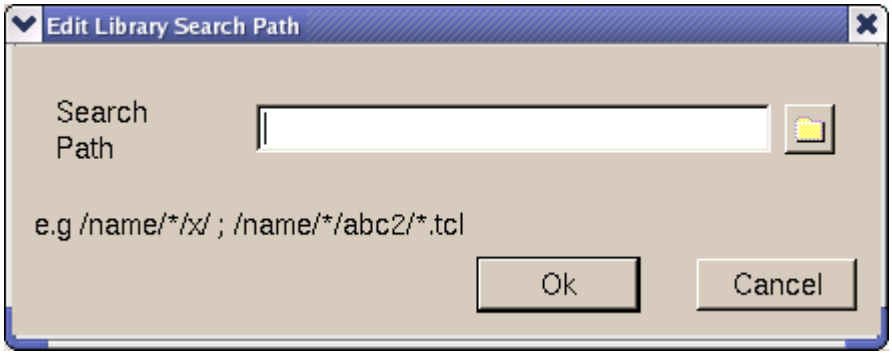
Remove From Library

This menu option is used to remove the information of the .xml file from the Library Browser. Choosing this option does not removes the file from the disk.

Edit Search Path

This menu option is used to specify the path of any .xml file of a component or an instance and include it in the Library Browser of the GUI. The following window opens when you select this menu option:

Figure 11-26 Edit Library Search Path



In the Search Path field, you can specify the directory path(s) of the file(s) that need to be included. If you need to specify more than one path, you can separate the paths using a semicolon (;).

Note that in the above field, you can use regular expressions, such as *, ?, and so on, at any level of the file hierarchy.

Scan Directory

This menu option is used to populate the library browser with all the .xml files in the chosen directory.

When you select this menu option, Choose a Directory to be Scanned window appears. Select the directory that needs to be scanned and click the OK button.

Generator Menu

When you click the Generator menu, the following options appear:

View Generated Files	Template

Generate VHDL	Generate Verilog

Note:
To know in detail about how to use generators, refer to *GenSys Generating Output Guide*.

Note:
GenSys saves the output of a generator in the output/<generator-name> directory.

View Generated Files

This menu option is used to display all the files generated in the previous run of the generators. This option opens a window displaying links to all files generated in the last run generator.

Template

This menu option is used to run Template generator to generate a report for a design or its components currently loaded in the GUI. The template generator uses a given template to generate the report.

Generate VHDL

This menu option generates the connectivity information for design currently loaded in GUI. This generates the RTL for the design in VHDL format.

GenSys displays the generated files in the GenSys Generator Reports window.

Generate Verilog

This menu option generates the RTL for the design in the Verilog format.

GenSys displays the generated files in the GenSys Generator Reports window.

Tools Menu

When you click the Tools menu, the following options appear:

Connection Builder	Configurability/Visibility	Elaborate
Incremental Schematic	Setup Rules	Interface Matching Widget

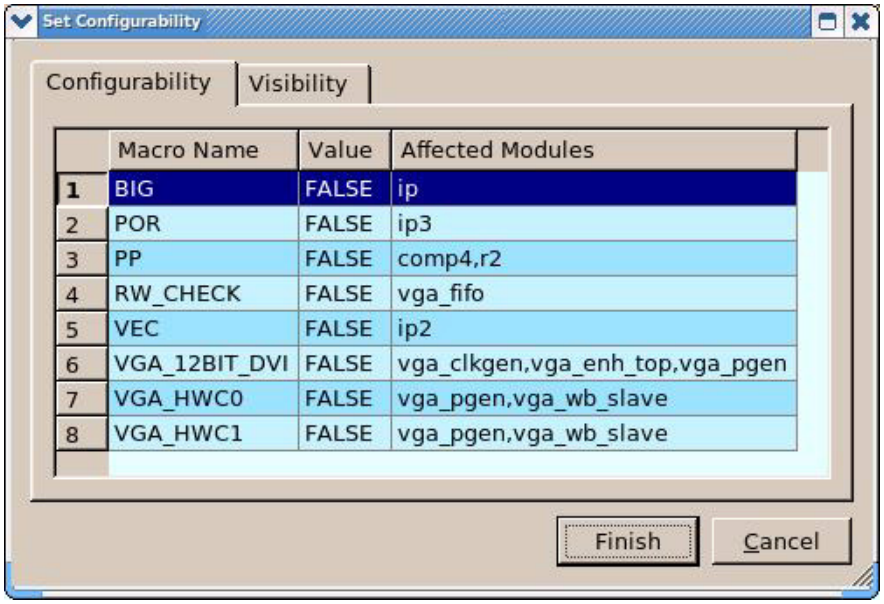
Tcl Editor Hierarchy Manipulation Options

Connection Builder

This menu option displays the Connection Builder dialog where you can configure and manage connections for the active design. To know more about managing connections using the Connection Builder dialog, see Managing Connections section.

Configurability/Visibility

This menu option displays the Set Configurability dialog, as shown in the following figure:



Use the above dialog to specify configurability and visibility at a global level.

To configure macros at the instance level, use the Configurability tab of the Property Editor pane. For details, see Using the Configurability Tab.

Elaborate

This menu option is used to elaborate all the connections of the design.

Incremental Schematic

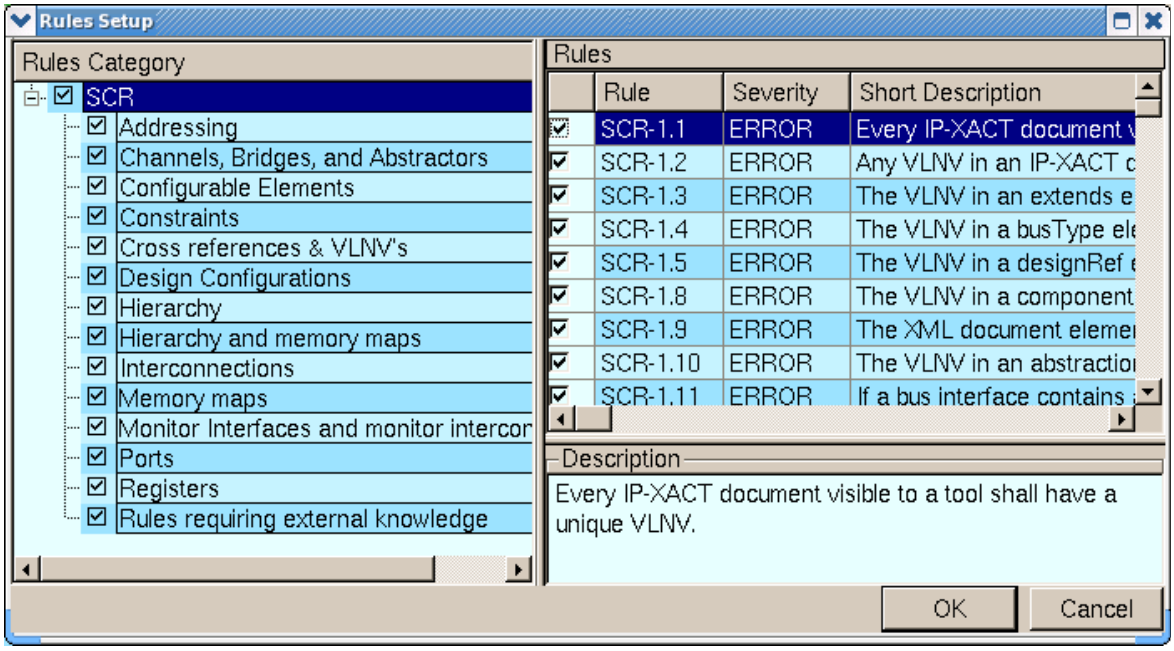
This menu option is used to display the Incremental Schematic window of the selected design instances. For details on the Incremental Schematic window, see Incremental Schematic Window section.

Setup Rules

This menu option displays the Rules Setup dialog that displays various GenSys rules that are reported at various stages in a particular GenSys session.

The following figure shows the Rules Setup dialog:

Figure 11-27 Rules Setup dialog



In the above dialog, the Rules Category section lists different categories of rules. When you select a category in this section, the corresponding rules of that category are displayed in the Rules section.

You can use the above dialog to:

- Enable or disable a rule to specify whether that rule should run or should not run, respectively.

  For example, in the dialog shown in Figure 11-27, if you want to disable the *SCR-9.2* rule, de-select the check box adjacent to that rule.

  You can also disable all rules of a particular category by de-selecting the check box adjacent to that category. For example, in the dialog shown in Figure 11-27, if you want

to disable all rules of the Addressing category, deselect the check box adjacent to this category.

- Change the severity of a rule

  If you want violations of a particular rule to be reported under a different severity node, you can change the severity of that rule.

  To change the severity of a rule, select the cell under the Severity column for that rule and select the required severity from the drop-down list.

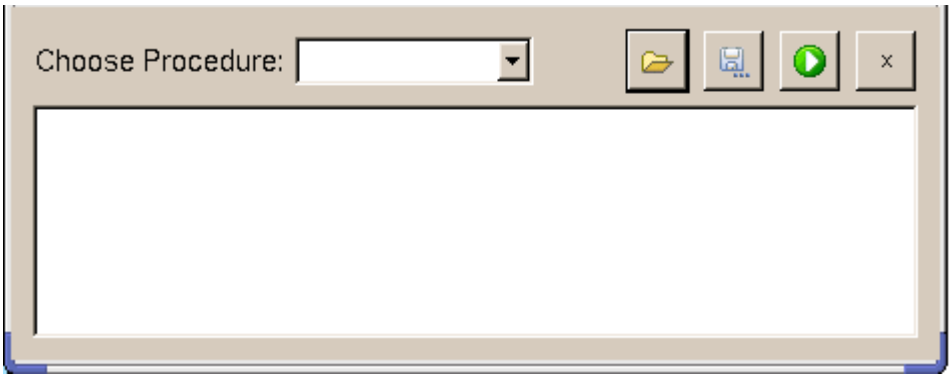
Interface Matching Widget

This menu option displays the Interface Matching Widget dialog. Use this dialog to map actual ports of a component with interface ports. For details, see Mapping Interfaces.

Tcl Editor

This menu option displays a blank Tcl Editor window, as shown in the following figure:

Figure 11-28 Tcl Editor Window



In the above window, you can create various procedures that can be used to perform different functions in GenSys. You can also edit the existing procedures by first loading an existing procedure by using the Open File button (

) and then making the required updates.

This dialog contains the following buttons:

Button	Description
	When you select this button, GenSys displays the Choose File dialog. In this dialog, select the required Tcl file containing the procedure that you want to display in the Tcl Editor dialog, and click the Open button. This displays the selected Tcl file in the Tcl Editor dialog. You can also replay the commands in the Tcl log file by selecting a Tcl log file. Refer to the *TclLog* and Specifying Commands sections.
	When you select this button, GenSys displays the Choose File dialog in which you need to select the Tcl file in which you want to save the changes. Once you select the required Tcl file, click the Save button.
	When you click this button, GenSys runs the Tcl procedure specified in text editor
	When you select this button, GenSys closes the Tcl Editor window.

Hierarchy Manipulation

This menu option displays the *Hierarchy Manipulation* dialog.

This dialog enables you to shift multiple IPs across the design hierarchy and compare the original design hierarchy with the new hierarchy before applying the hierarchy changes.

Using this dialog, you can perform the following tasks:

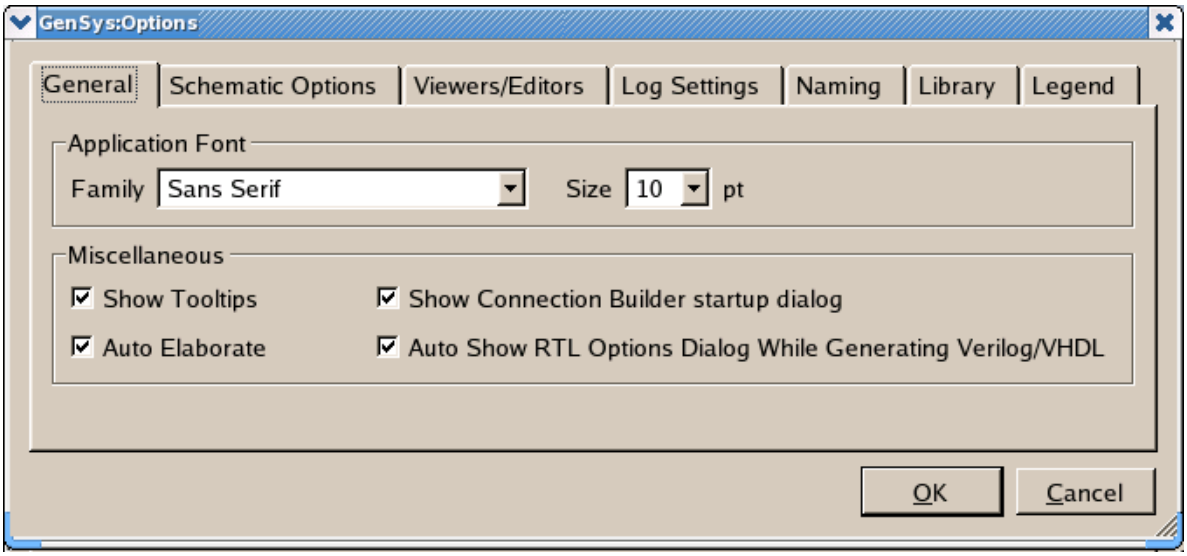
- Adding a Hierarchy
- Dissolving a Hierarchy
- Grouping
- Viewing Library Cells in a Hierarchy
- Comparing Original and New Hierarchies
- Moving Instances

- Cloning Instances
- Adding Clones in the Hierarchy of Instances of the Specified Design Masters
- Solidifying Glue Instances and Generate Blocks

Options

This menu option displays the Options dialog where you can configure various options under different tabs.

Figure 11-29 Options Dialog



The following points discuss details of various tabs of the above dialog:

- The General tab

  This tab contains various check boxes to specify settings, such as enabling/disabling automatic elaboration, display of tool tips in the schematic, showing Connection Builder startup dialog, and RTL options.

  To enable/disable automatic elaboration, select/de-select the Auto Elaborate check box. When you select this check box to enable automatic elaboration, the elaboarate_glue and elaborateIO Tcl commands do not refresh glue hookups instantly in the schematic. A complete elaboration of the design is needed for that. Run the elaborate Tcl command instead to view all glue hookups refreshed in the schematic window.

- The Schematic Options tab

  This tab contains options to control color and miscellaneous settings in the schematic. For details, see Updating Color Settings and Miscellaneous Settings.

- The Viewer/Editors tab

This tab is used to set the path of viewers for various generators. You can specify the path for the PDF reader, xml viewer, text editors, Word file reader, HTML viewer, and description editor.

- The Log Settings tab

  This tab is used to set the command history size and maximum log lines.

- The Naming tab

  This tab is used to set the naming format and start index of interfaces and instances. You can specify the Suffix/Prefix that should be attached to instance or interface names created in GenSys. You can also set the start index (integer value) for instance and interfaces in GenSys.

- The Library tab

  This tab is used to set the File Type Priority for XML, Tcl, and IP-XACT files. Selecting XML in the drop-down list will give priority to XML files while loading new files in the GUI.

  You can also check the Enable TCL Autoload field to enable autoload of Tcl files.

- The Legend tab

  This tab is used to change the colors of the area under Library Browser and Design Browser. It is also used to specify a different color for cells containing invalid values.

Reports Menu

When you click the Reports menu, the following options appear:

Design Summary	Interface Summary
Connection Summary	
Diff Report	Diff Report (Details)
Check RTL	RTLHC Report
Packaging Report	ERC Report

Note:
You can also generate the above reports by using the create_report Tcl command. For details on this Tcl command, refer to *GenSys Tcl Commands Reference Guide*.

Specifying a Directory to Save Reports

By default, all the above reports are generated under the directory specified by the set_report_dir Tcl command. For example, specify the following command to save all reports in the /reports directory:

```
set_reports_dir ./reports
```

Alternatively, you can specify a different directory by clicking the Save As button appearing in the report and specifying the desired location.

Design Summary

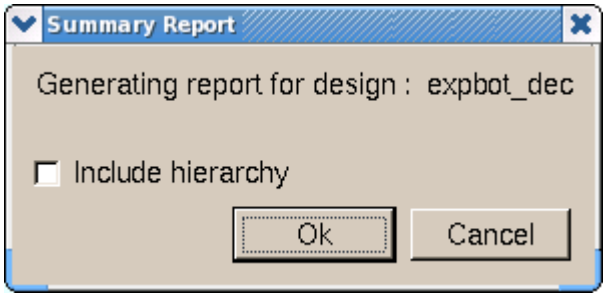
This menu option is used to generate a summary report that contains the following details:

- Summary of the entire design
- Details of instances at each level of the design and parameters that are passed from an instance hierarchically
- Summary of the number of adhoc connections, tieoff connections, open connections, and interface connections.
- Total number of unconnected pins/ports at each level

  Note:
  A design object must be active to generate the design summary report.

When you select this menu option, the Summary Report dialog appears for the selected design/component.

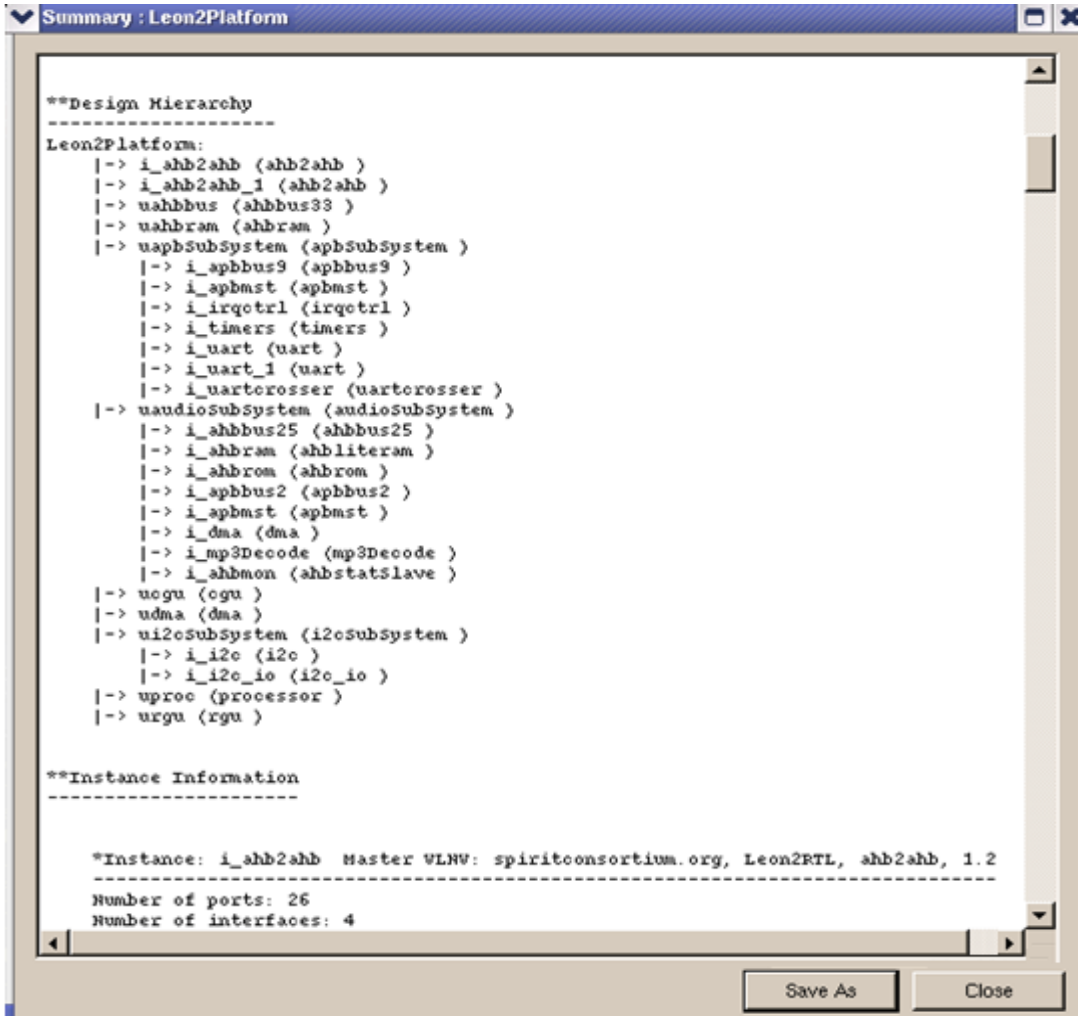
For example, the following figure displays this dialog for the selected design expbot_dec:



In the above dialog, if you select the Include hierarchy check box, the report displays a summary of the subsystems also instantiated in the selected design/component.

When you click the OK button in the Summary Report dialog:

- A design summary report of the name <design-name>_summary.txt is created in the specified reports directory.
- Contents of the report appears in a separate window, as shown in the following figure:



Interface Summary

This menu option is used to generate an interface summary report that contains the following details:

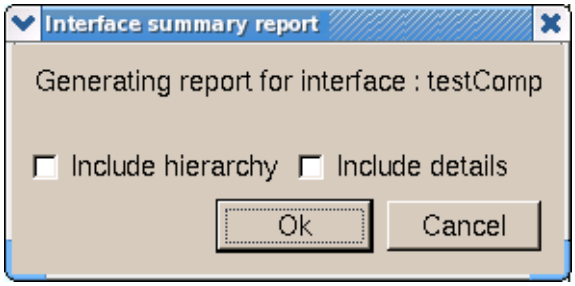
- A summary of interfaces instantiated in a component or a design
- Details, such as interface names, type of interfaces, interface mapping of actual ports to logical ports for each instantiated interface in a component/design

- Percentage mapping of logical ports to actual ports. This is very useful during IP packaging as it gives a first glimpse of unmapped ports in each interface.
- For a design, the report shows a list of instances and interfaces instantiated for a particular instance. A hierarchical view of the report provides the same information for each hierarchy of the design.

  Note:
  A design object or a component object must be active to generate the interface summary report.

When you select this menu option, the Interface Summary Report dialog appears for the selected design/component.

For example, the following figure displays this dialog for the selected component testComp:



The above dialog contains the following check boxes:

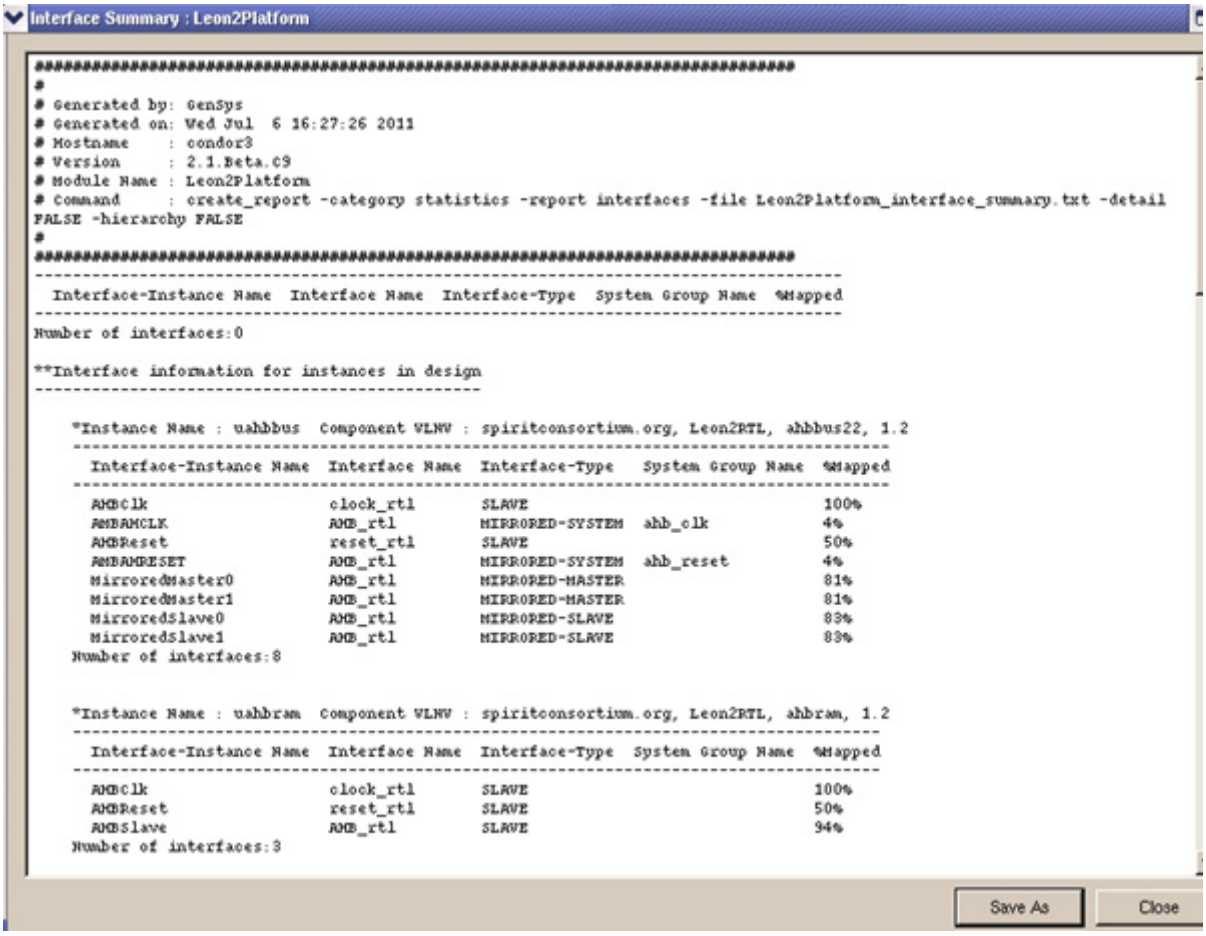
- Include hierarchy

  Select this check box to include interface summary for each design/component object across the design hierarchy.

- Include details

  Select this check box to include additional details, such as logical to actual port-mapping details.

The following figure shows an example of the Interface Summary Report:



```
Interface Summary : Leon2Platform
##############################################################################
#
# Generated by: GenSys
# Generated on: Wed Jul  6 16:27:26 2011
# Hostname    : condor3
# Version     : 2.1.Beta.C9
# Module Name : Leon2Platform
# Command     : create_report -category statistics -report interfaces -file Leon2Platform_interface_summary.txt -detail
FALSE -hierarchy FALSE
#
##############################################################################
-------------------------------------------------------------------------------
  Interface-Instance Name  Interface Name  Interface-Type  System Group Name  %Mapped
-------------------------------------------------------------------------------
Number of interfaces:0

**Interface information for instances in design
-----------------------------------------------

    *Instance Name : uahbbus  Component VLNV : spiritconsortium.org, Leon2RTL, ahbbus22, 1.2
    -------------------------------------------------------------------------------
      Interface-Instance Name  Interface Name  Interface-Type   System Group Name  %Mapped
    -------------------------------------------------------------------------------
      AMBClk                   clock_rtl       SLAVE                               100%
      AMBAMCLK                 AMB_rtl         MIRRORED-SYSTEM  ahb_clk            4%
      AMBReset                 reset_rtl       SLAVE                               50%
      AMBAMRESET               AMB_rtl         MIRRORED-SYSTEM  ahb_reset          4%
      MirroredMaster0          AMB_rtl         MIRRORED-MASTER                     81%
      MirroredMaster1          AMB_rtl         MIRRORED-MASTER                     81%
      MirroredSlave0           AMB_rtl         MIRRORED-SLAVE                      83%
      MirroredSlave1           AMB_rtl         MIRRORED-SLAVE                      83%
    Number of interfaces:8

    *Instance Name : uahbram  Component VLNV : spiritconsortium.org, Leon2RTL, ahbram, 1.2
    -------------------------------------------------------------------------------
      Interface-Instance Name  Interface Name  Interface-Type  System Group Name  %Mapped
    -------------------------------------------------------------------------------
      AMBClk                   clock_rtl       SLAVE                              100%
      AMBReset                 reset_rtl       SLAVE                              50%
      AMBSlave                 AMB_rtl         SLAVE                              94%
    Number of interfaces:3
```

Save As Close

Connection Summary

This menu option is used to generate a connectivity report that provides a summarized view of the entire design connectivity.

This report contains the following details:

- Total number of connections
- Number of adhoc connections
- Number of interface connections
- Number of exported interface connections
- Number of tie-off connections
- Number of intentional open

- Number of unconnected scalar terms

In addition to each instances in a design at the current level, this report shows connection information (adhoc-in, adhoc-out and interface) of other instances at the same level.

Note:
A design object must be active to generate the connectivity report.

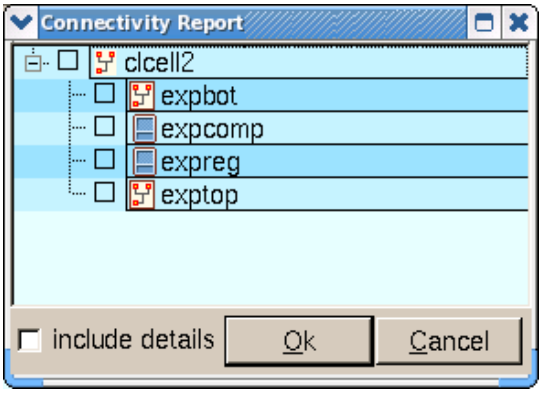
Viewing Connections Summary of the Selected Subsystem

To view connection summary of a subsystem, perform the following steps:

1. Load a design by double-clicking the required design in the Library Browser Tab.
2. Select the Report -> Connectivity Report menu option.

   The Connectivity Report dialog appears. For example, the following figure displays this dialog for the design clcell2:

Figure 11-30 Connectivity Report



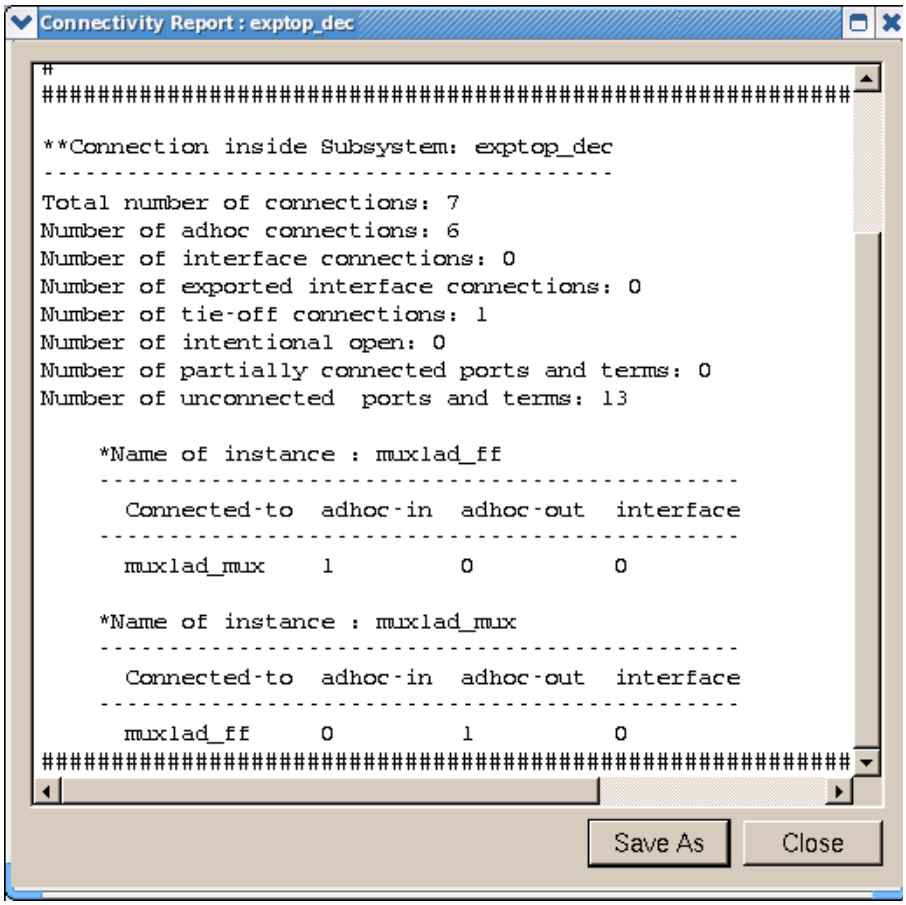
The above dialog shows the top-level design and subsystem designs present in the design hierarchy.

1. In this dialog, select the required subsystem for which you want to view connection summary.
2. Click the OK button to close the above dialog.

After performing the above steps, the Connectivity Report appears displaying the connection summary of the selected subsystem.

This report shows details, such as total number of different type of connections in the selected subsystem and connection summary of various instances within that subsystem.

For the example, the following figure shows the Connectivity Report for the subsystem exptop:



Viewing Additional Connection Details of an Instance within the Selected Subsystem

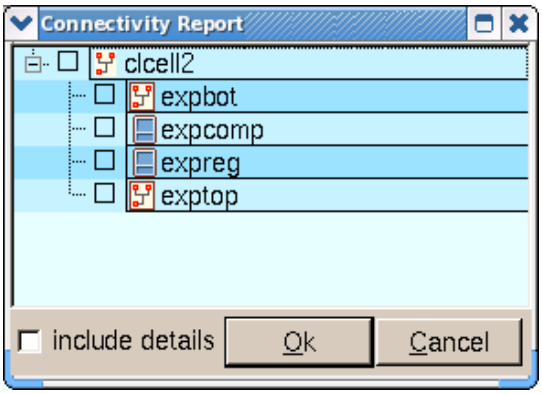
In addition to the connections summary of the selected subsystem, you can also view additional connection details of an instance within that subsystem.

To view these details, perform the following steps:

1. Load a design by double-clicking the required design in the Library Browser Tab.
2. Select the Report -> Connectivity Report menu option.

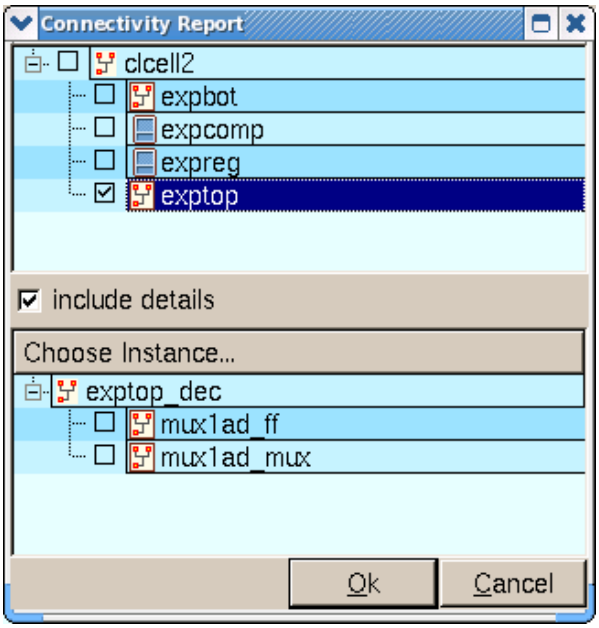
   The Connectivity Report dialog appears. For example, the following figure displays this dialog for the design clcell2:

Figure 11-31 Connectivity Report



3. In the above dialog, select the required subsystem for which you want to view connection summary.
4. Select the Include details check box in the above dialog.

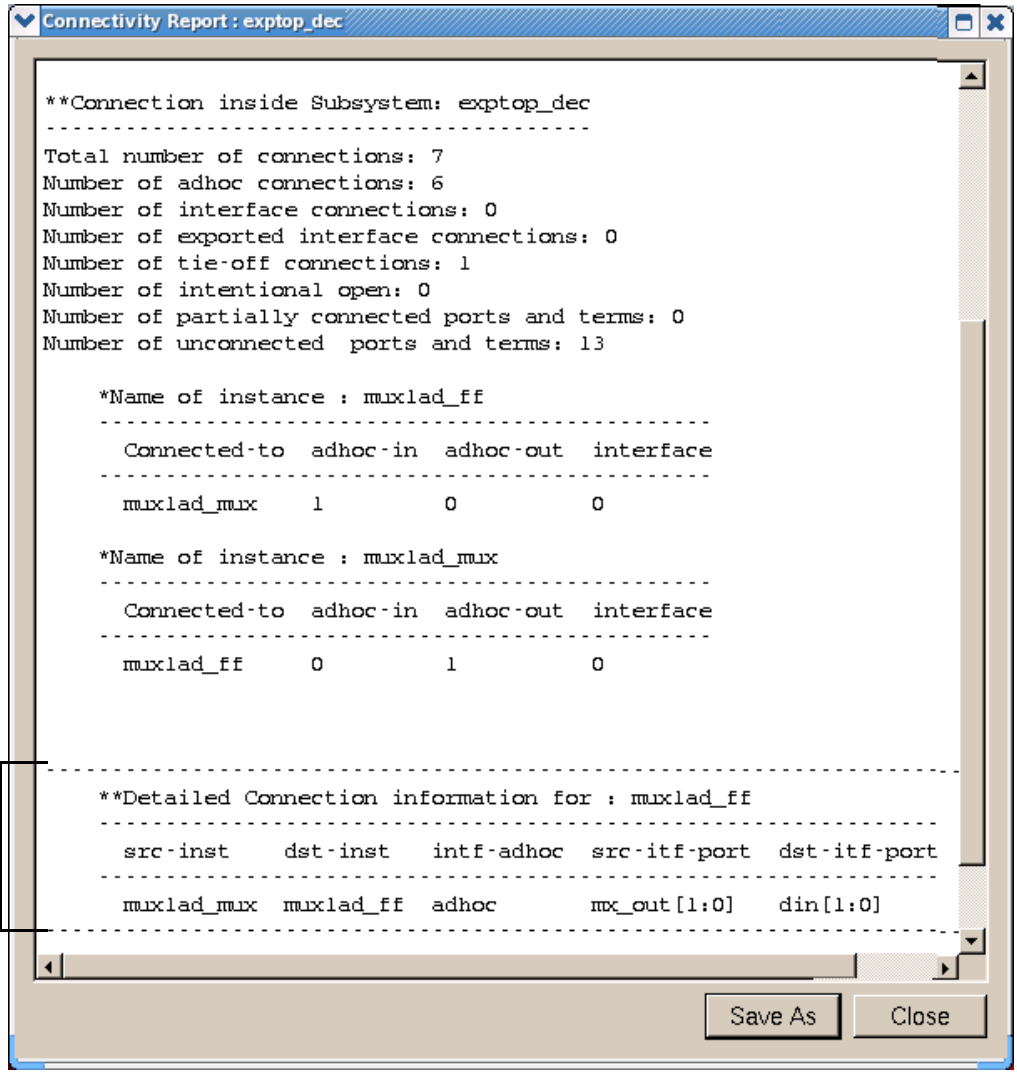
   This displays the Choose Instance section in the Connectivity Report dialog, as shown in the following figure:



5. From the Choose Instance section in the above dialog, select the required instance for which you want to view additional connection details.
6. Click the OK button to close the Connectivity Report dialog

After performing the above steps, the generated Connectivity Report will contain additional connection information for the selected instance in addition to connection summary of the selected subsystem.

For example, the following figure shows the additional connection information for the selected instance, mux1ad_ff, in addition to connection summary of the selected subsystem:



Additional connection information of the selected instance mux1ad_ff

Suggested Interfaces

This menu option is used to generate the Suggested Interfaces Report that displays details of interfaces suggested by the tool for the design and components.

Diff Report

This menu option is used to generate the design difference report. This report shows the following details:

- Differences in ports, parameters, interfaces, and attributes between two designs or components.
- Differences in instances and connectivity for designs only.
- Differences in filesets for components only.

To use the design difference report, read the following topics:

- Use Models for the Design Difference Report
- Information in the Design Difference Report
- Generating the Design Difference Report
- Intrepreting the Design Difference Report

To generate the detailed design difference report, refer to Diff Report (Details).

Use Models for the Design Difference Report

Some of the use-models where the design difference report is useful are:

- Update Master: Use this report to know the difference between two IPs before updating them. You can detect whether the changes are acceptable.
- Concurrent Engineering: Use this report to detect changes between various checked-in revisions of the SoC when multiple designers are working together in an SoC assembly flow.
- Equivalence Check: Use this report to know if the version of the IP being assembled and the latest RTL version are equivalent since multiple RTL drops occur while the top-level is being stitched in an SoC assembly flow.

  In addition, when the Library Browser has multiple copies of the same design or component, it might be difficult to determine which object to select even after viewing the VLNV of the object. By using this report, you can identify key differences to aid in the object selection.

Information in the Design Difference Report

The design difference report compares the current design/component with a reference design/component. The information compared are described in the following table.

Items	Fields Compared
Port	Direction, MSB, LSB, Port Type, HDL Port Type, Interface Mapping, HDL Pragma, Default value associated with ports, Visibility(Conditionality)
Parameter	Type, Value, RTL, Const
Interface	Definition Name, Vendor, Version, Library, Type, isMirrored, Group
Attributes	Type, Value
Instances	Master Name, Master Vendor, Master Version, Master Library, Parameter Info (name and value)
Filesets	Group, Dependency, Files
Connectivity	SrcInstance (Source Instance), SrcPin(Source Pin), DstInstance(Destination Instance), DstPin(Destination Pin)

Generating the Design Difference Report

You can generate the design difference report by:

- Using Tcl Commands
- Using the GenSys GUI

Using Tcl Commands

Use the following command to generate the design difference report:

```
create_report -category differences
```

When you run the above command, GenSys creates the <currentVLNV>_difference.txt file under the directory specified by the set_report_dir Tcl command.

For further details, refer to the Tcl Commands Reference Guide.

Using the GenSys GUI

There are multiple ways to generate the Design Diff report from the GenSys GUI.

- You can generate the Design Diff report even if no object is loaded in GenSys. Select the Diff Report option from the Report menu. The Select Design to Diff dialog box displays the designs that are currently in the library. Select the two designs you want to compare.
- For designs/components that are loaded in GenSys:
  - Select two designs in the library browser, right-click menu, select the Show Diff Report option.
  - Alternatively, select the Diff Report option from the Report menu. The Select Design to Diff dialog box displays the designs that are currently in the library. Select a design. GenSys generates the Design Diff report by comparing the selected design/component with the active design/component.

Lastly, you can generate the Design Diff report by performing the following steps:

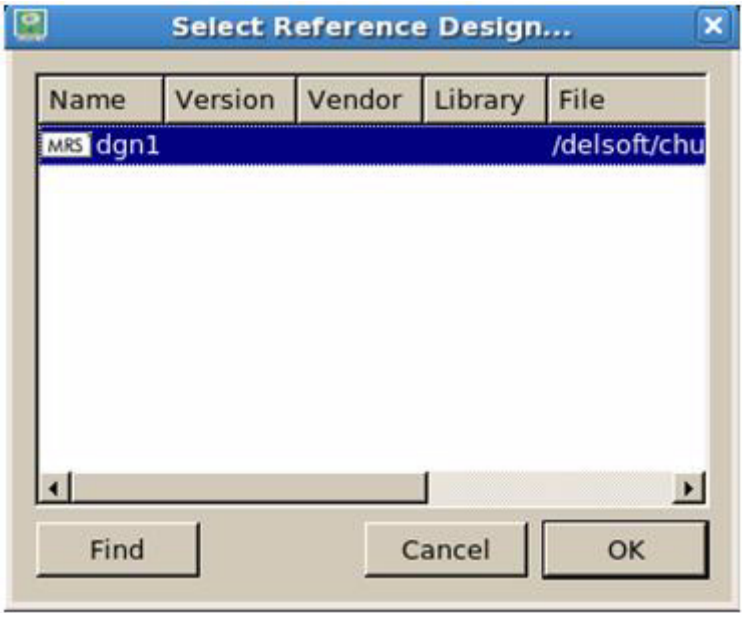
1. Select the design/component that you want to treat as the current version.

   This version will be compared with a Reference design or component.

2. Select the Diff Report option from the Report menu to select the Reference design/component.

   The Select Reference Design dialog box appears, as shown in Figure 11-32.

Figure 11-32 The Select Reference Design Dialog Box

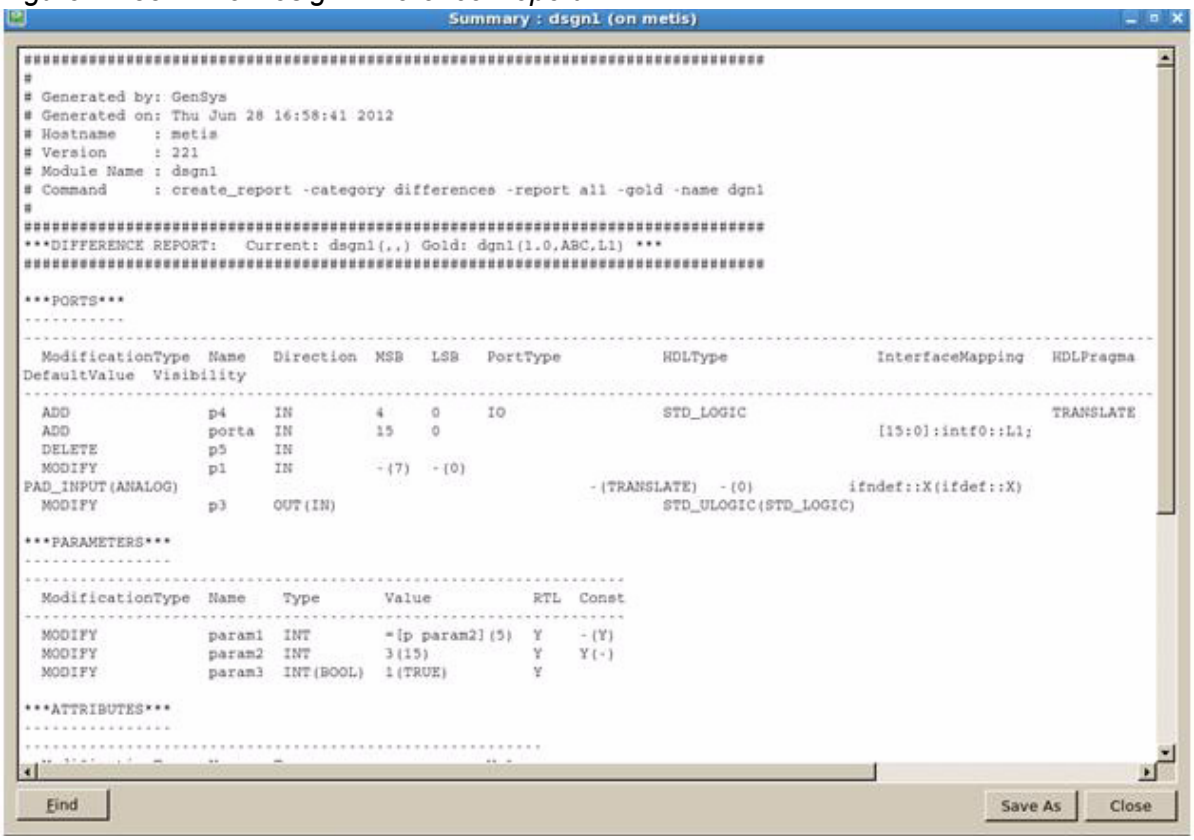


The above dialog box lists the designs/components that are available in the library.

3. Select the reference design/component, and click the OK button.

The design difference report appears in a new window, as shown in Figure 11-33.

Figure 11-33 The Design Difference Report



Intrepreting the Design Difference Report

The design difference report represents differences in the current component/design with respect to reference component/design by identifying the following modification types:

- ADD: Items with this modification type are not in the reference design/component, but are present in the current. For example, the P4 port has been added in the current in the current design, as shown in Figure 11-34.
- DELETE: Items with this modification type are present in the reference design/component but are not present in the current design/component. For example, the P5 port exists in the reference design/component, but does not exist in the current design/component, as shown in Figure 11-34.
- MODIFY: Items with this modification type are in the reference design/component with the same name as the current. However, the other fields, such as type and value of the item, are modified in the current.

In the fields that are modified, GenSys reports the current data followed by the data in the reference design/component in brackets.

If either the current or golden data is empty, it is represented by a dash "-". For example, the P1 port is modified. The current design does not contain a value for MSB and LSB. However the reference design has an MSB value of 7 and an LSB value of 0, as highlighted in the following figure.

Figure 11-34 Interpreting the Design Difference Report

```
***PORTS***
-----------
-------------------------------------------------------------------------------------------------------------------------------
  ModificationType  Name    Direction  MSB   LSB   PortType          HDLType                InterfaceMapping   HDLPragma
DefaultValue  Visibility
-------------------------------------------------------------------------------------------------------------------------------
  ADD               p4      IN         4     0     IO                STD_LOGIC                                     TRANSLATE
  ADD               porta   IN         15    0                                              [15:0]:intf0::L1;
  DELETE            p5      IN
  MODIFY            p1      IN         -(7)  -(0)
PAD_INPUT(ANALOG)                                        -(TRANSLATE)  -(0)          ifndef::X(ifdef::X)
  MODIFY            p3      OUT(IN)                          STD_ULOGIC(STD_LOGIC)
```

Diff Report (Details)

The design difference report only shows comparisons of items that are added, deleted, or modified. If you need detailed comparison information on any item, view the detailed design difference report.

Since this report uses the TkDiff utility, ensure it is installed in your system.

To generate the detailed design difference report, perform the following steps:

1. Select the design/component that you want to treat as the current version.

   This version will be compared with a Reference design or component.

2. Select the Diff Report (Details) option from the Report menu to select the reference design/component.

   The Select Reference Design dialog box appears. The dialog box lists the designs/components that are available in the library.

3. Select the reference design/component, and click the OK button.

   The Select Items for Diff dialog appears, as shown in the following image.

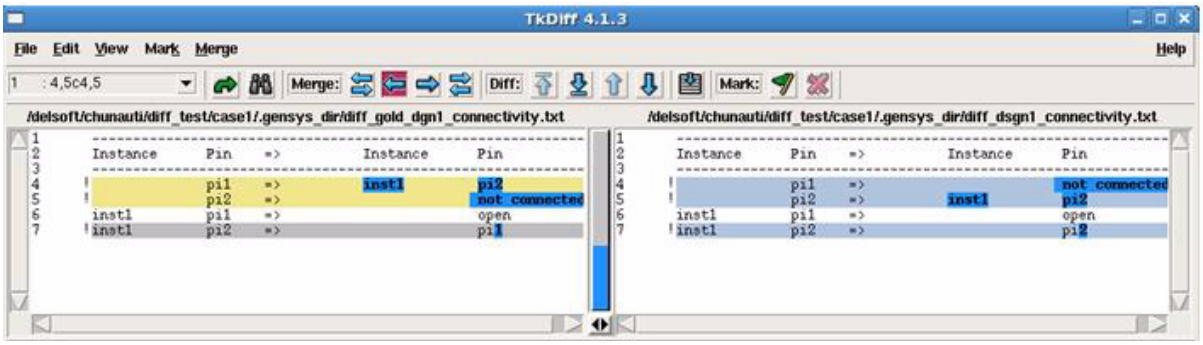
4. Select the required check box. For example, the Connectivity check box is selected in Figure 11-35.

Figure 11-35 The Select Items for Diff dialog box

5. Click the OK button.

   The detailed design difference report for connectivity appears in the TkDiff utility, as shown in Figure 11-36.

Figure 11-36 The Detailed Design Difference Report in TkDiff



If you select multiple check boxes, such as Ports, Parameters, and Filesets in the Select Items for Diff dialog, GenSys generates separate text reports for the current and reference component/design containing information about the items you selected for comparison. The TkDiff utility generates the report by comparing these two text files.

These text reports for the current and reference component/design are stored in .gensys_dir/ inside the current working directory. The following naming convention is used:

For current design/component files, the naming convetion for the text report is:
diff_<current vlnv>_<item type>.txt

For reference design/component files, the naming convention for the text report is:
diff_gold_<gold vlnv>_<item type>.txt

Packaging Report

The GenSys IP packaging sub-flow creates an IP-XACT view from the RTL view of the IP. This report provides a summary of the characteristics of the packaged IP view.

This report takes as an input the VLNV of an IP or works on the active component.

Type of Information Reported

Some of the information reported in this report are:

- VLNV of the IP
- Ports properties
- Interfaces and mapping details
- Parameter details, including conditionality
- Filesets of the IP

How to Generate the Packaging Report

You can generate the report for a component or a design. If the report is generated for another type of object, such as an interface, an error message is displayed. For a component, the report is generated immediately. However, when the report is generated on a design, a design browser dialog appears.

You can select a design or component instance from the hierarchy contained within the open design. The report is then run on that node of the design.

You can generate this report in any of the following ways:

- By using the Report -> Packaging Report menu option in GUI.
- By using the create_report_Packaging Tcl command.

```
create_report_Packaging [-file <file-name>]  -help -currentoptions
-resetoptions -noexec -triggerOn {list} -triggerOff {list}
```

Refer to the *GenSys Tcl Commands Reference Guide* for more information on this Tcl command.

ERC Report

The ERC report works on the selected or active design. It reports various DRC (design rule check) violations in the current or selected design. The DRC violations are reported at current hierarchy level and not at flattened level.

Type of Information Reported

Some of the information reported in this report are:

- Floating Terminals: This helps in quickly identifying the unconnected terminals or those terminals which are connected with floating wires. This report will take as additional inputs, -input to only report input ports which are hanging and -output to report output ports which are hanging. The default behavior would be to report all the input and output floating terminals
- Multiple Drivers: This report provides data for all the terminals which are multiply driven.
- Partial Vector Connections
- Fully Connected Vector Connections: Analyzed for pin re-arrangement
- Fully Connected Vector Connections: Analyzed for pin re-arrangement by splicing
- Only multiple drivers on the design level are reported. It will not recursively iterate the hierarchy.

How to Generate the ERC Report

You can generate the report for a design. If the report is generated for another type of object, such as an interface, an error message is displayed. When the report is generated on a design, a design browser dialog appears.

The options may also be selected at the Tcl prmpt with -<option name> <option value>. For custom reporting, these options are sticky within a run.

You can generate this report in any of the following ways:

- By using the Report -> ERC Report menu option in GUI.
- By using the create_report_ERC Tcl command.

```
create_report_ERC [-disconnects <TRUE|FALSE>]  [-input <TRUE|FALSE>]
[-output <TRUE|FALSE>]  [-opens <TRUE|FALSE>]  [-partials
<TRUE|FALSE>]  [-endians <TRUE|FALSE>]  [-splices <TRUE|FALSE>]
[-tieoffs <TRUE|FALSE>]  [-ports <TRUE|FALSE>]  [-verbose
<TRUE|FALSE>]  [-file <filename>] [-multidrivers <TRUE|FALSE>] -help
-currentoptions -resetoptions -noexec
```

Refer to the *GenSys Tcl Commands Reference Guide* for more information on this Tcl command.

RTLHC Report

This menu option generates information about the connectivity of terminals in an instance and ports of a design. For details of this report, refer to the Checking the Design Health section.

Check RTL

The Check RTL report is useful for Checking the RTL Against the Packaged View. This report provides a comparison of loaded RTL components with the component with the same VLNV from library browser. The report is saved as check_rtl.log located in the report directory.

To learn more about this report, refer to the following sections.

- Type of Information Reported
- How to Generate the Check RTL Report
- Intrepreting the Check RTL Report

Type of Information Reported

The comparison is performed for:

- Ports
- Parameters
- Filesets

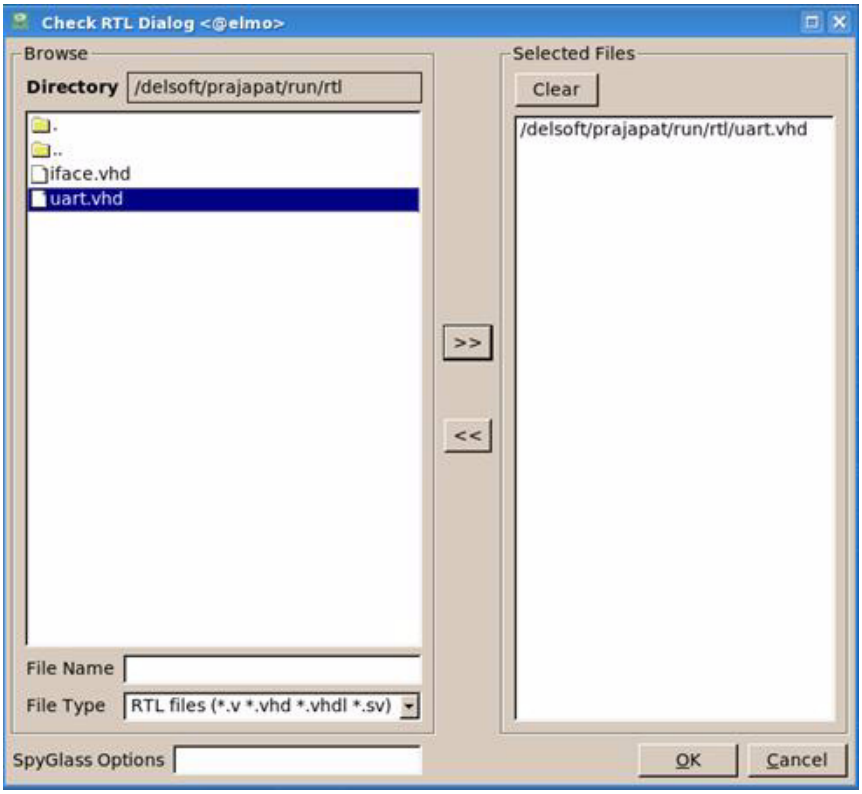
How to Generate the Check RTL Report

You can generate this report in any of the following ways:

- By using the check_rtl Tcl command. Refer to the *GenSys Tcl Commands Reference Guide* for more information on this Tcl command.
- By using the Report -> Check RTL menu option in the GUI.

To generate the report by using the GUI, perform the following steps:

1. Click Reports -> Check RTL. The Check RTL dialog box appears.
2. Browse to the RTL file in the left pane and the click the >> button. The RTL appears in the Selected File pane.



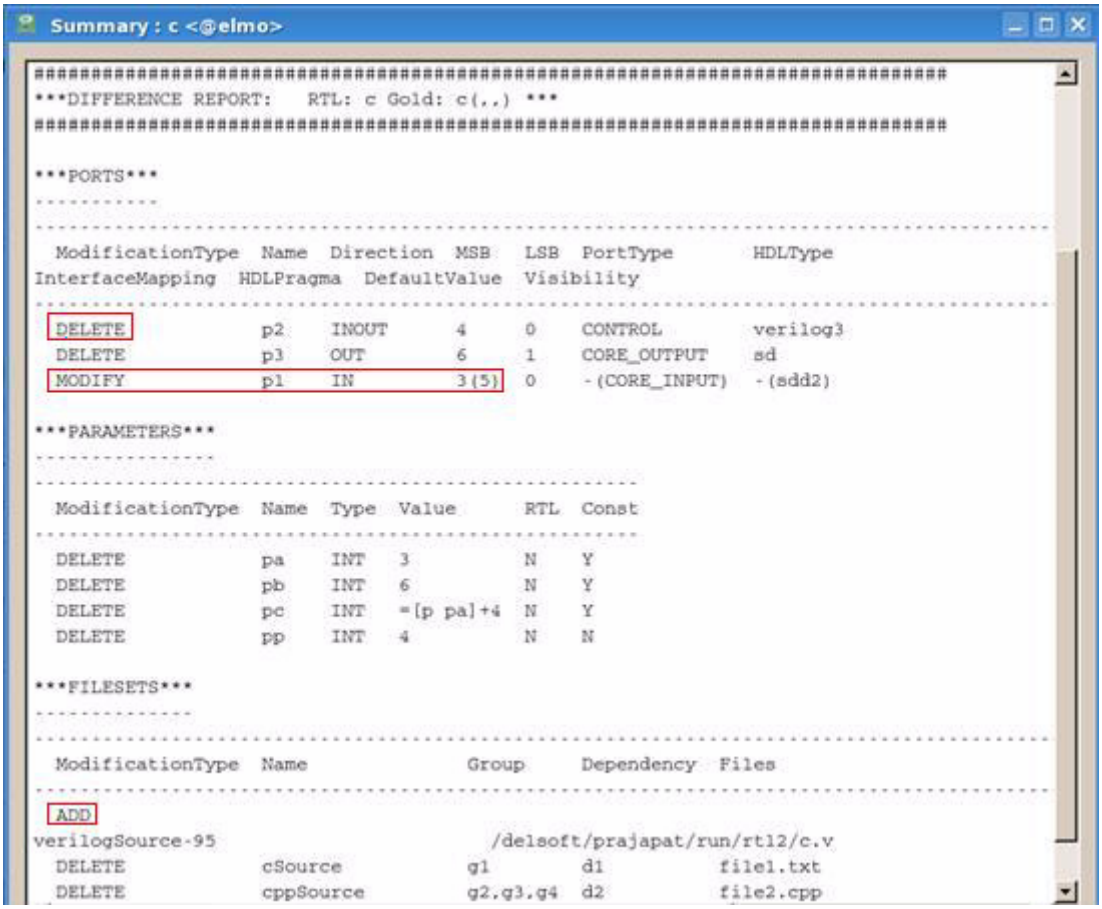
3. Click OK. The Check RTL report appears in a new window.

```
Summary : c <@elmo>
################################################################################
***DIFFERENCE REPORT:   RTL: c Gold: c(,,) ***
################################################################################

***PORTS***
-----------
--------------------------------------------------------------------------------
  ModificationType  Name  Direction  MSB   LSB  PortType       HDLType
InterfaceMapping  HDLPragma  DefaultValue  Visibility
--------------------------------------------------------------------------------
  DELETE            p2    INOUT      4     0    CONTROL        verilog3
  DELETE            p3    OUT        6     1    CORE_OUTPUT    sd
  MODIFY            p1    IN         3(5)  0    -(CORE_INPUT)  -(sdd2)

***PARAMETERS***
----------------
-----------------------------------------------------
  ModificationType  Name  Type  Value      RTL  Const
-----------------------------------------------------
  DELETE            pa    INT   3          N    Y
  DELETE            pb    INT   6          N    Y
  DELETE            pc    INT   =[p pa]+4  N    Y
  DELETE            pp    INT   4          N    N

***FILESETS***
--------------
--------------------------------------------------------------------------------
  ModificationType  Name              Group     Dependency  Files
--------------------------------------------------------------------------------
  ADD
verilogSource-95                        /delsoft/prajapat/run/rtl2/c.v
  DELETE            cSource           g1        d1          file1.txt
  DELETE            cppSource         g2,g3,g4  d2          file2.cpp
```

Intrepreting the Check RTL Report

The Check RTL report provides comparison of loaded RTL components with the component with the same VLNV from the library browser by identifying the following modification types:

- ADD: Items with this modification type are not in the reference design/component, but are present in the current.
- DELETE: Items with this modification type are present in the reference design/component, but are not present in the current.
- MODIFY: Items with this modification type are in the reference design/component, but have a different associated value in the current. The modification is reported The current data is reported followed by the data in the reference design/component in brackets. For example, p1 has a value of 5 in the reference design/component and a value of 3 in the current, as highlighted in the following figure.



Help Menu

When you click on the Help menu, following options appear:

GenSys User Guide	Tcl Commands Reference	Generating Output Guide
Customization Guide	Schema Guide	Assembly Methodology Guide
Registers Methodology Guide	Generating IO Logic Guide	Release Notes
KPNS	About	

GenSys User Guide

This menu option is used to display *GenSys User Guide*.

Tcl Commands Reference

This menu option is used to display the *GenSys Tcl Commands Reference Guide*.

Generating Output Guide

This menu option is used to display the *GenSys Generating Output Guide*.

Customization Guide

This menu option is used to display the *GenSys Customization Guide*.

Schema Guide

This menu option is used to display the *GenSys Schema User Guide*.

Assembly Methodology Guide

This menu option is used to display the *GenSys Assembly Methodology Guide*.

Registers Methodology Guide

This menu option is used to display the *GenSys Registers Methodology Guide*.

Generating IO Logic Guide

This menu option is used to display the *GenSys Generating IO Logic Guide*.

Release Notes

This menu option is used to display the *GenSys Release Notes* document that contains details about all enhancements made in the current GenSys version.

KPNS

This menu option is used to display the *GenSys KPNS* document that contains details about known problems in GenSys and their corresponding solution.

About

This menu option is used to check the current GenSys version.

Toolbar

The GenSys toolbar has the following buttons:

Button	Description
	The New toolbar button is used to create a new design, interface, or a component. Select the required option and click the OK button.
	The Open toolbar button is used to open an existing design, interface, or component. Select the file that needs to be opened and click the OK button.
	The Save toolbar button is used to save the changes made to a design, interface or component. Select the required destination directory where the design file needs to be saved and click the OK button.
	The Save All toolbar button is used to save the changes made to all the open files.
	The Modular Schematic button is used to display the schematic for the selected design.
	The Incremental Schematic button is used to display the Incremental Schematic of the selected instance in the design.
	The Options button is used to display the Options dialog to configure Viewer Settings, Log Settings, Naming Settings, General Settings, and Library Settings.

Button	Description
	The Insert Row toolbar button is used to insert a new row with default values after the current selected row in a table.
	The Insert Sub Table toolbar button is used to insert a subtable in the selected row (for example, bitfields for a register, bitEnumValues for a given bitfield).
	The Delete Row toolbar button is used to delete the selected row in the Data Table window.
	The Zoomout Table Font button is used to zoom out the table font size.
	The Zoomin Table Font button is used to zoom in the table font size.
	The Goto Previous Error button is used to go to the previous error shown after validation.
	The Goto Next Error button is used to go to the next error shown after validation.
	The Sync IPXACT and table view button is used to sync up the IP-XACT Viewer pane with the changes made in IP-XACT table.

Button	Description
	The Setup Rules button is used to open the Rules Setup dialog that displays various GenSys rules that are reported at various stages in a particular GenSys session.
	The Interface Matching Widget button is used to open the Interface Matching Widget dialog that is used to map actual ports of a component with interface ports and map rules to interfaces.

12

Using GenSys in the Batch Mode

Following are the command-line options provided in GenSys:

```
gensys
```

- [-batch]
- [-autortl2mrs <files> -dir <destdir>]
- [-mrs2autortl <files> -dir <destdir>]
- [-l <dirName>]
- [-schema <dirName> | <file>]
- [-includeAutoRTLSchema
- [-vendorExtnSchema <vendor-name>]
- [-useSupplyName]
- [-batchSch]
- [-dumptcl <files> -dir <destdir>]
- [-dumpxml <file> -dir <destdir>]
- [-dumpspirit <file> -dir <destdir>]
- [-dumpsingletcl]

- [-debug]
- [-log_dir <dir>]
- [--perflog]
- [-outdir <dir>]
- [-XML_DIR <dir>]
- [-report_dir <dir>]
- [-tcl <files>]
- [-help]
- [-stopTclInterpOnErrors]
- [-strict]
- [-trace]
- [-use_old_elab_flow]
- [-version]
- [-64bit]
- [-syncSpirit]
- [-dumpsingletcl <files> -dir <destdir>]
- [-product]
- [-rcfile <config-file>]
- [-nologLink]

-batch

Use this option to invoke GenSys in the batch mode.

If you do not specify this option, GenSys starts in the GUI mode.

-autortl2mrs <files> -dir <destdir>

Use this option to convert the specified AutoRTL files into MRS files in the directory.

-mrs2autortl <files> -dir <destdir>

Use this option to convert the specified MRS files into AutoRTL files in the directory.

-I <dirName>

Use this option to add the directory to the GenSys Perl include path.

-schema <dirName> | <file>

Use this option to read the .schema files located in the specified directory or the specified schema file.

-includeAutoRTLSchema

Use this option to read the AutoRTL .schema files also.

-vendorExtnSchema <vendor-name>

Use this option to read the vendor-specific .schema files also.

-useSupplyName

Use this option to change the Verilog RTL code containing 1’b0 and 1’b1 in the RTL design to VSS and VCC, respectively. For example, the following code

```
assign sig = 1'b0
```

will be replaced with

```
assign sig = VSS
```

Similarly, 1'b1 will be changed to VCC.

-batchSch

Use this option to generate schematic in the batch mode using Tcl commands. To know details of the Tcl commands, refer to the *Tcl Commands Reference Guide*.

-dumptcl <files> -dir <destdir>

Use this option to generate Tcl files for the loaded XML files in the destination directory.

-dumpxml <file> -dir <destdir>

Use this option to generate XML files corresponding to the loaded files in the directory. This option is used for IP-XACT to XML conversion.

-dumpspirit <file> -dir <destdir>

Use this option to generate IP-XACT files corresponding to the loaded files in the directory. This option is used for IP-XACT to SPIRIT conversion.

-dumpsingletcl

Use this option to generate one Tcl file corresponding to the specified xml files. By default, the file is dumped in the current working directory. You can specify a different directory by using the -dir command-line option.

-debug

Enables the display of debug/processing messages in the Message Log window of GenSys GUI. If you do not provide the -debug option, only error and warning messages are shown in the GUI; debug/processing messages are not displayed.

-log_dir <dir>

Use this option to specify a directory where GenSys log files will be created. Working directory is the default log directory.

--perflog

Use this option to create a file, gensys.perflog, in the log directory. GenSys logs the details of execution of Perl functions in this log file. This file contains a log of time elapsed and memory consumed every time GenSys calls a Perl function, such as generator functions or validation functions. In addition, the total time taken for multiple runs of a Perl function in the current session is also logged. Memory usage is given in terms of megabytes being consumed at a given point of time. This information helps in checking the performance of Perl functions.

-outdir <dir>

Use this option to specify the output directory <*dir*>. All new XML files and internal generator output files will be created in the output directory.

Working directory is the default output directory.

-XML_DIR <dir>

Use this option to specify the directory where new XML files will be created. Output directory is the default XML directory.

-report_dir <dir>

Use this option to specify the reports directory. Reports include files created by internal generators.

Working Directory is the default reports directory.

-tcl <files>

Use this option to specify Tcl files that should be run in GenSys. The location of the argument specification is not essential. If you specify the argument without associating it with any specific command, then the argument is considered as Tcl files for the -tcl option.

-help

Use this option to print the usage information. It lists all command-line options along with their purpose.

-stopTclInterpOnErrors

Use this option to stop the processing of Tcl command in the specified Tcl file, if there are Tcl command errors in the file.

-strict

Use this option to report stack trace of the perl call in case of errors in execution of the perl function. By default, GenSys reports only the line of error.

-trace

Use this option to log Info messages in gensys.log file. By default, error and warning messages are logged in gensys.log file. The -trace option also enables logging of Info messages.

-use_old_elab_flow

Use this option to make connection elaboration non-order based. This also enables the old splice connection behavior that creates normal interface connection for non-spliced bit and logical ports of an interface.

Note:
This option provides backward compatibility for old designs. It is not recommend to use this option for new designs.

-version

Use this option to view GenSys version.

-64bit

Use this option to run GenSys on 64-bit Linux platform.

-syncSpirit

Use this option to generate spirit extension tables with designs.

-dumpsingletcl <files> -dir <destdir>

Use this option to generate a Tcl file corresponding to each XML file specified in argument in the specified directory. The Tcl files have the same name as that of the corresponding XML file.

Consider an example in which you specify the following command:

```
gensys -batch -dumpsingletcl resetval.xml
```

The above command creates a single file, resetval.cmd.

-product

Use this option to invoke the required GenSys product. You can specify any of the following argument with this command:

Argument	Description
assembly	Invokes the GenSys Assembly product.
io	Invokes the GenSys IO product.

Argument	Description
power_plan	Invokes the GenSys Power Planning product.
power_reconc	Invokes the GenSys Power Reconciliation product.
registers	Invokes the GenSys Register Solution product.

You can invoke multiple products at the same time by specifying this command multiple times with different product names.

Note:
If you do not specify any product by using the -product command-line option, GenSys tries to invoke the Assembly product.

-rcfile <config-file>

Use this option to specify a configuration file (~/.gensys/gensysrc) that contain various user specified settings.

Whenever you make any changes in the GUI, such as hiding tabs/columns, GenSys generates a configuration file that contain the details of these settings. This configuration file can be shared among various users. To use this configuration file, specify the file name with this command-line option.

Note:
It is recommended not to modify this file manually.

-nologLink

Use this option to ensure that a link to the log file should not be created in the home area after running GenSys.

GenSys Batch Mode Specific Options

Following command-line options can be used only when running GenSys in the batch mode (when -batch option is used to invoke GenSys).

```
gensys
  -batch
```

[-gen pdf <files> <destdir>]

[-gen sequence

[-seqfiles [file | {<module>=<file>}]]

[{ -I <dirName> <dir> }

<files> *<destdir>*]

[-gen rtl <verilog | VHDL> *<design-name>* *<perlfile>* *<files>* *<destdir>*]

[-usergen *<generator-name>* [*<xml-files>*]]

[-stop_untouched_hierarchy]

-gen pdf <files> <destdir>

Use this option to generate PDF files for the specified generator in the *<destdir>* directory.

-gen sequence

Use this option to generate sequence files. The -seqfiles switch specifies a single sequence file. The <module=file> option specifies the sequence file for the respective modules. The -I switch is used to specify the include paths.

-gen rtl

Use this option to generate RTL for the design *<design-name>* according to the connectivity specified by the Perl file. The design can be in Verilog or VHDL language.

-usergen

Use this option to run user-defined generator *<generator-name>*. You may also specify the XML files on which the specified generator *<generator-name>* needs to be run.

-stop_untouched_hierarchy

When this option is specified, GenSys parses all design modification commands specified in Tcl files and populates a list of modified hierarchies. During RTL design read specified in the command-line Tcl files, GenSys applies stop to all hierarchies, which are not present in the list of modified hierarchies populated earlier.

For example:

```
gensys -batch -stop_untouched_hierarchy -tcl test.tcl
```

13

Working with Registers

Registers are an integral part of any IP. They can be of varied bit sizes, such as 16, 32, 64, or even larger bit sizes. Registers can be written into and read out by different data buses. In addition, some of the bits can be reserved, status, or can write-only bits.

Managing register configurations for various registers and for different IPs is a challenge. The complexity increases when design functionality changes. This leads to modification of registers. As a result, registers are often out of sync with design specification and documentation.

GenSys provides a mechanism to interactively capture different register definitions for various IP's and generate necessary C-header files to enable early software development, RTL files in either Verilog or VHDL, and corresponding documentation.

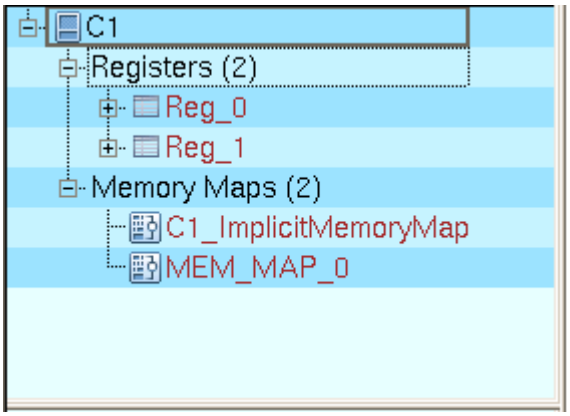
To use and take advantage of the *Register* product, you need a separate license key. After you have the license key, specify the -product register option while invoking GenSys.

This chapter provides information on how to use GenSys while working with registers and memory maps.

Capturing Component Register Information Using GUI

When you load a component under the Design Browser, GenSys displays the details of registers and memory maps of that component under the component hierarchy.

An example is shown in the following figure:



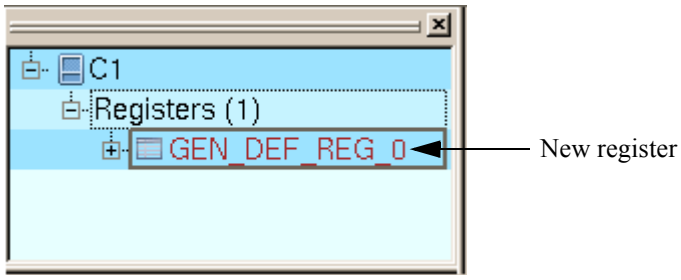
Specifying Register Definitions

To add register for a component, select the component name under the Design Browser, and perform any of the following actions:

- Select Tools -> Registers/Memory Map -> New Register menu option.
- Select the New Register button (

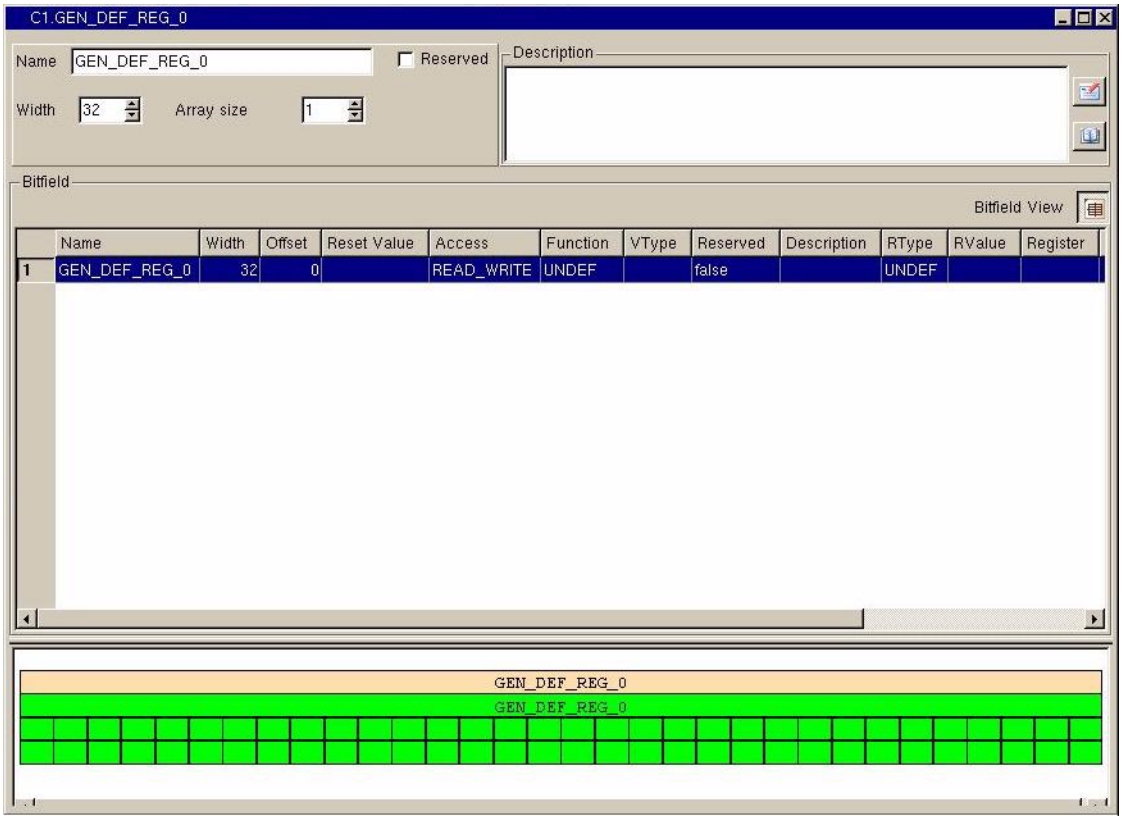
-) from the toolbar.

On performing any of the above actions, a new register (with a default name) appears under the component under the Design Browser, as shown in the following figure



In addition, the following window appears:

Figure 13-1 Editing



In the above window, you can change the default name of the newly added register in the Name field. In addition, specify register details, such as width, array size, and description in appropriate fields.

To add more registers, perform any of the following actions:

- Right-click on the Registers text and select Add Register option from the short-cut menu.

- Right-click on the register name and select Add Register option from the short-cut menu.
- Select Tools -> Registers/Memory Map -> New Register menu option.
- Select the New Register button (

-) from toolbar.

On performing any of the above actions, an editor window appears in which you can specify the required details.

Note:
You can also add register information in the HW_SW.RegisterBlock table of the component or by using the add_register Tcl command.

Specifying Register Description

You can specify a description for a register in the Description field.

To add a description, perform the following steps:

1. Click the Edit Description button (

2.) adjacent to the Description field.

   A default editor window appears.
3. Specify the description in the editor window.
4. Click the OK button to close the editor window.

The specified description appears in the Description field.

You can also import another file containing the description. To import a file, perform the following steps:

1. Click the Import Description File button (

2.) adjacent to the Description field.

   The Specify the file containing description dialog appears.
3. Select the required file from this dialog.

4. Click the Open button.

The description containing in the selected file gets dumped in the Description field.

Specifying Reset Information

While adding register details under the HW_SW.RegisterBlock table, if you do not know the reset value, you can specify reset value as x or X. The value x or X will be treated as unknown and while register file generation, macros for these reset values will not be dumped in the generated header (*.h) file.

Modifying Register Details

To modify register details, double-click on a register. This displays an editor window in which you can make the required updates.

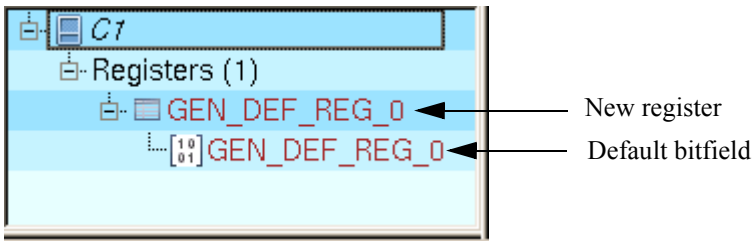
You can undo or redo any change made in this editor dialog by selecting the Undo (

) and Redo (

) buttons, respectively, from the toolbar in GenSys main window.

Specifying Bitfield Definitions

By default, GenSys adds a default bitfield for each newly added register. An example is shown in the following figure:



The name of the default bitfield is same as the name of the register, width same as that of the register, and access type as READ/WRITE.

To add more bitfields for a register, perform any of the following actions:

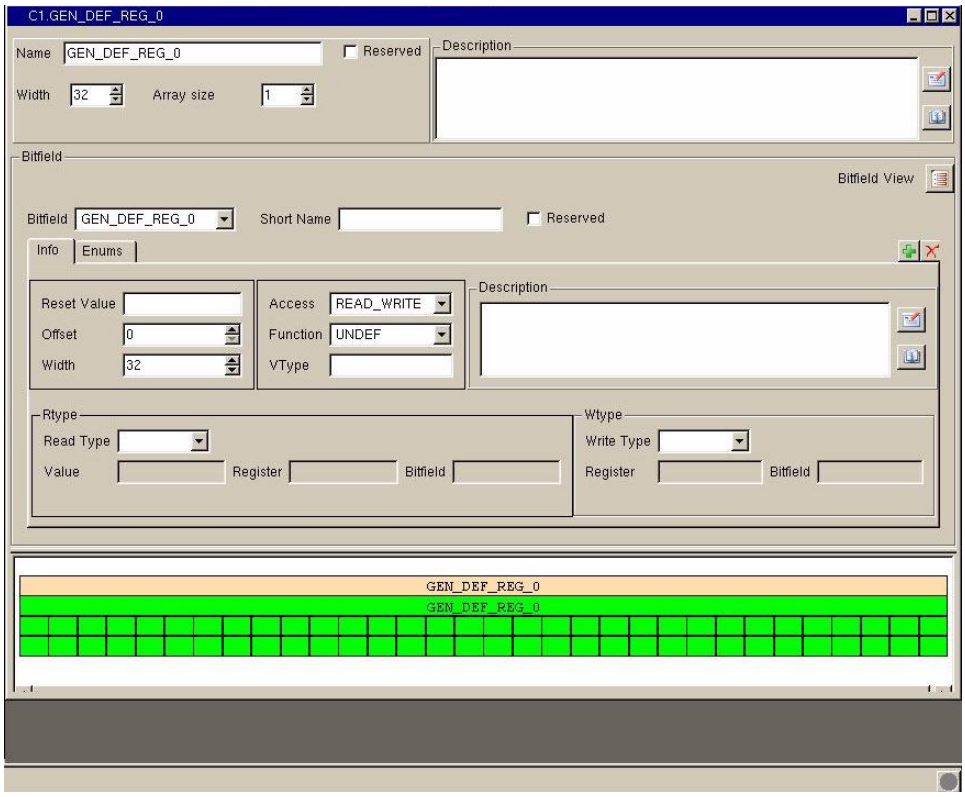
- Right-click on a register appearing under the Design Browser and select the Add Bitfield option from the short-cut menu.
- Right-click on a bitfield appearing under the Design Browser and select the Add Bitfield option from the short-cut menu.
- Select Tools -> Registers/Memory Map -> New Bitfield menu option.
- Select the New Bitfield button (

-) from the toolbar.
- Click

- in the editor window displayed for a register.

  This displays the Add Bitfield dialog in which you can specify the name of the new bitfield.

On performing any of the above actions, the following window appears



:

In the above window, you can specify bitfield details, such as Bitfield Short Name, Reset Value, Offset, Access, Vtype, Width, and Function under the Bitfield section.

In addition, you can specify the rtype, wtype of a bitfield by selecting a value from the drop-down list. Depending upon the rtype and wtype selected from this drop-down list, GenSys displays additional fields in which you need to specify the required value. The following table specifies the additional fields displayed by GenSys when you select a particular value in Read Type and Write Type drop-down lists:

Field	**Value selected from drop-down list**	**Additional Field(s) Displayed**
RType	Rreturns	Value
	RreturnsRemote	Register Bitfield

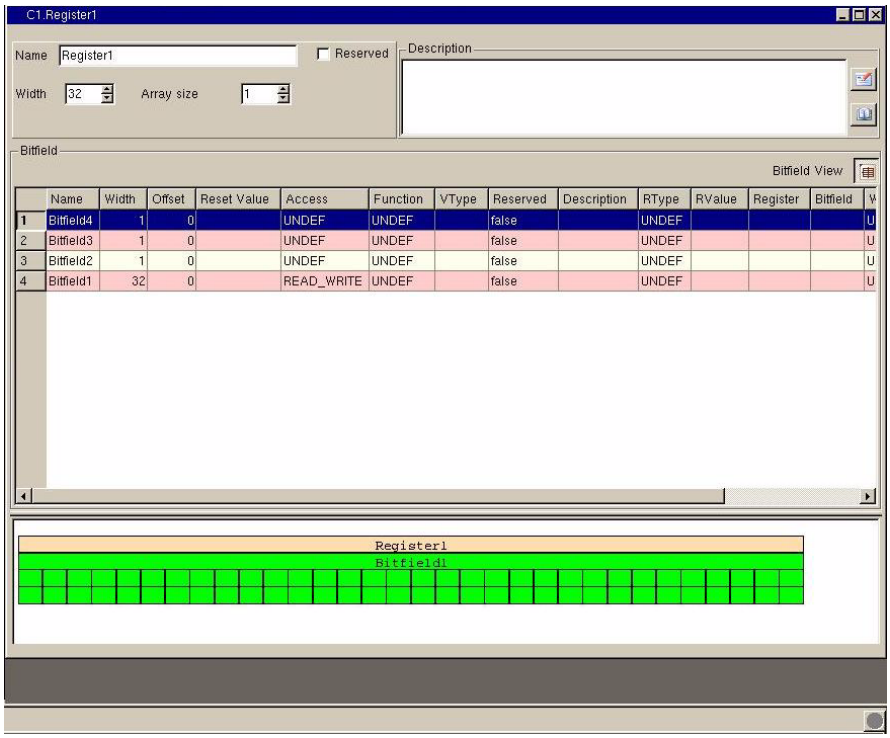
Field	Value selected from drop-down list	Additional Field(s) Displayed
WType	WtoClr	Register Bitfield
	W0toClr	Register Bitfield
	W0toSet	Register Bitfield
	W1toSet	Register Bitfield

You can also view the details of all the added bitfields for a register. to do so, click the Table View button (

). This displays all the bitfield information in a tabular format. Alternatively, to view the information in form view, click the Form View (

) button.

An example of the tabular view is shown in the following window:



Note:

You can also add register information in the HW_SW.RegisterBlock.bitfields table of the component or by using the add_bitfield Tcl command.

Modifying or Deleting Bitfield Definitions

To modify a bitfield definition, double-click on the bitfield appearing under the Design Browser. This displays an editor window in which you can make the required updates.

To delete a bitfield definition, select the bitfield appearing under the Design Browser and click

.from the toolbar.

Alternatively, if you are using the editor window, from the perform the following steps to delete a bitfield:

1. Double-click on a register appearing under the Design Browser.

   An editor window appears.

2. Click

3. in the editor window to switch to the form view of bitfields.
4. Select the bitfield to be deleted from the Bitfield drop-down list.
5. Click

6. to delete the selected bitfield.

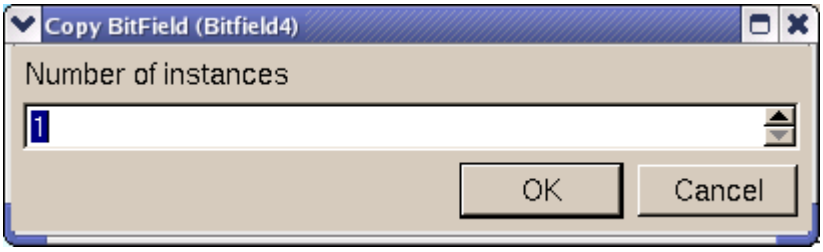
Copying Bitfield Definitions

If you wish to use same bitfield definition(s) present in a register, you can copy the required bitfields from that register.

To copy a bitfield of a register, perform the following steps:

1. Right-click on the bitfield to be copied.
2. Select the Copy or Cut option from the shortcut menu, depending upon your requirement.
3. Right-click on the register in you want to add that bitfield.
4. Select the Paste option from the shortcut menu.

   This displays the Copy Bitfield dialog, as shown in the following figure:



5. In the above dialog, specify the number of instances to be added for the bitfield being copied.

Interpreting the Type of a Bitfield

You can interpret the type of a bitfield based on its access type. Refer to the following table to determine the type of bitfield based on its access type:

Bitfield Access Type	Bitfield Type
Execute, Read, Read Clear, Read Execute	READ
Read Write, Read Write Clear, Read Write Exec, Read Write Set, Write One To Set	READWRITE
Reserved, Undef, Non Accessible, Program Memory	OTHERS
Write, Write Clear, Sticky	WRITE
Unused	UNUSED

The types of bitfields in a register helps in determining the register type. The following table describes how register type can be determined based on the types of bitfields:

WRITE/ READWRITE Bitfield	READ Bitfield	OTHERS Bitfield	Register Type
1	1	X	READWRITE
0	1	X	READ
1	0	X	WRITE
0	0	X	OTHERS

Completing Bitfield Definitions

You may add some bitfields with some empty holes. In such cases, to complete the bitfield definition, perform the following steps:

1. Select the HW_SW.RegisterBlock table.
2. Right-click on the row of the register for which you want to complete bitfield definitions.
3. Select the Complete BitField option from the shortcut menu.

If you right-click on the table, holes in all the registers of the component are filled. However, if you right-click on a particular row (register), holes in only that register are filled.

For each empty hole, a new row is created in the bitfield subtable. Each new row has bitfield name as _RESV and bitfield type as RESERVED. These bitfields are also marked with the GENESIS_BITFIELD_DEFAULT attribute.

Alternatively, you can also use the complete_bitfield TCL command to fill the holes. To know more, refer to the *Tcl Commands Reference Guide*.

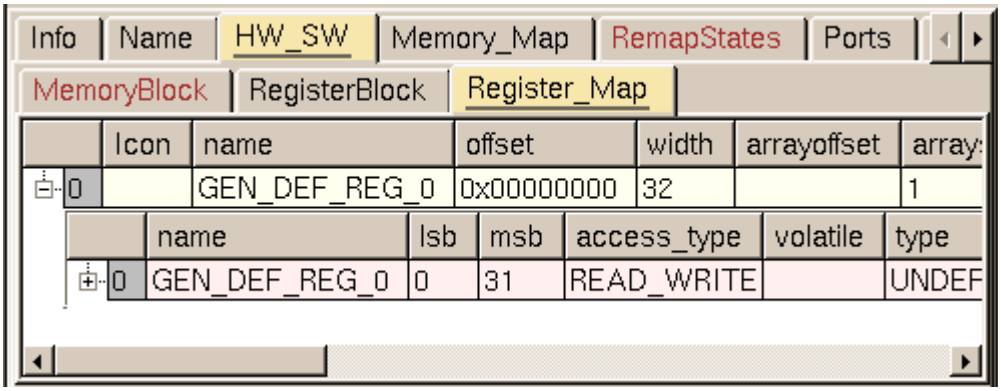
After completing the bitfields, if you make some changes in the bitfield definitions, you can again fill the empty holes by using either of the two ways.

Viewing Register Map

A register map gets created by using the information entered in the HW_SW.RegisterBlock table for the component.

To view the register map, double-click the component under the Design Browser, and select the Register_Map tab under the HW_SW tab. The register map information for the selected component gets displayed in the Data Table window, as shown in the following figure:

Figure 13-2 Register Map



If you do not specify an offset value for a register, it will be calculated (for generating register map view) using the default MAU of the component, the offset, and the width information of the previous register or group. In the register map, the offset value can start only from a multiple of MAU or from 0. The automatically generated value will be shown in the Register_Map table, but it will not be entered in the RegisterBlock table.

Note:

MAU is calculated from the default register width specified in the register_width column (given in bits) under the Info tab of a component. This value is converted into bytes while calculating MAU. If no value is specified in the register_width column for the component, the component default MAU value is considered as 1.

If two registers, *Register1* and *Register2*, are at the same offset, you might want to generate a warning and to give preference to any one of them. This decision to generate a warning and to select a register to which preference should be given depends upon the type of each register. This decision can be based on the following table :

Register1 type	**Register2 type**	**Conflict (Display warning?)**	**Preference**
READ	READ	YES	Register1
READ	WRITE	NO	Register2
READ	OTHERS	NO	Register1
READ	READWRITE	YES	Register2
WRITE	READ	NO	Register1
WRITE	WRITE	YES	Register1
WRITE	OTHERS	NO	Register1
WRITE	READWRITE	YES	Register1
READWRITE	READ	YES	Register1
READWRITE	WRITE	YES	Register1
READWRITE	OTHERS	NO	Register1
READWRITE	READWRITE	YES	Register1
OTHERS	READ	NO	Register1
OTHERS	WRITE	NO	Register1
OTHERS	OTHERS	NO	Register1
OTHERS	READWRITE	NO	Register1

Using the reset values entered for the bitfields of a register block, the reset value is shown in the resetval column of the Register_Map table. In case if some of the bitfields of a register have an unknown reset value, a partial bitfield information gets displayed in the resetval column of that register. Refer to the figure below:

	icon	name	offset	width	instaddr	numinst	resetval	resetmask
0		R1	0x00000000	32		1	0X0002XXXX	

	name	lsb	msb	access_type	description	resetval
0	b1	0	15	READ_WRITE		x
1	b2	16	31	READ_WRITE		2

1		R2	0x00000004	48		1	0XXXXX0002XXXX	

	name	lsb	msb	access_type	description	resetval
0	b1	0	15	READ_WRITE		x
1	b2	16	31			2
2	b3	32	47			x

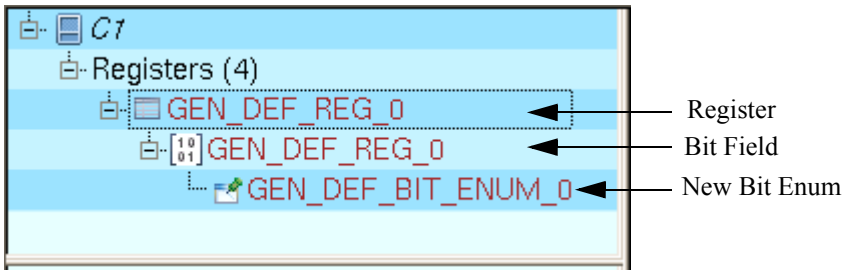
2		R5	0x0000000A	16		1	0X6XXX	

	name	lsb	msb	access_type	description	resetval
0	b1	0	9	READ_WRITE		x
1	b2	10	12			2
2	b3	13	15			3

In the above figure, the resetval column for register, R1, consists of partial bitfield information (0x0002xxxx).

Specifying Bit-Enums

To add a bitenum for a bitfield, right-click on a bitfield name appearing under the Design Browser and select the Add Bitenum option from the shortcut menu. A new bitenum then appears under the bitfield under the Design Browser. An example is shown below:

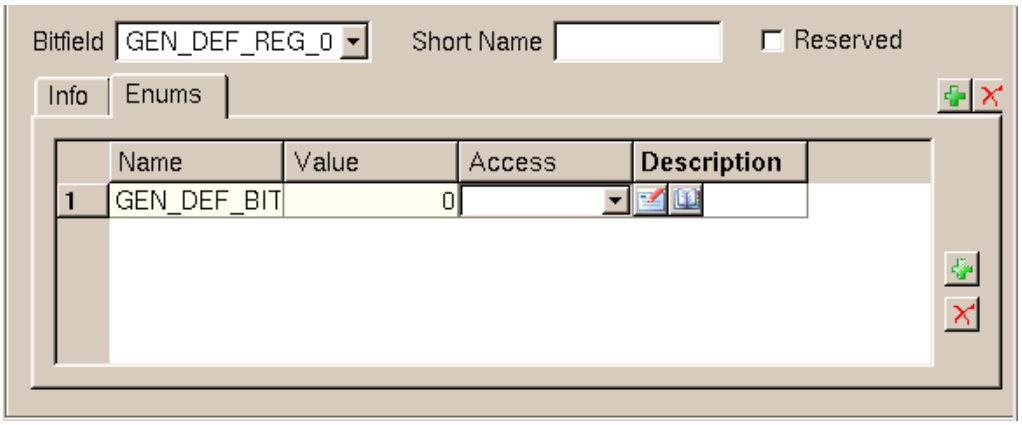


To add details for the bitenum, perform the following steps:

1. Double-click on the bit-enum appearing under the Design Browser.

   An editor window appears.

2. Click the Enums tab in the editor window. The following page appears:

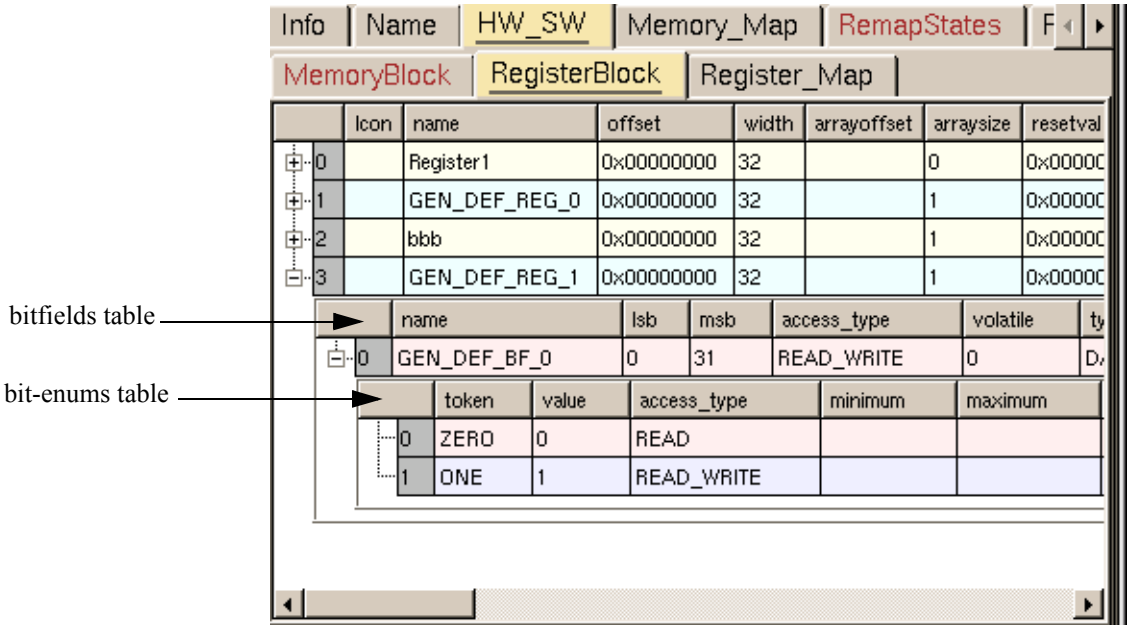


3. Specify the bit-enum details under the Enums tab.

To add more bit-enums, click

under the Enums tab. This adds a new row in which you can specify the details of another bit-enum.

You can also add bit-enums in the subtable of the HW_SW.RegisterBlock.bitfields table. An example is shown in the following figure:



You can copy and paste bit-enums across various bitfields. The process is same as copying and pasting bitfields. For details, see Copying Bitfield Definitions.

If you want to delete a bit-enum, select that bit-enum appearing under the Design Browser and click

.from the toolbar. Alternatively, if you are working in the editor window, select the bit-enum and click

.

Creating Register Groups

To create register groups, perform the following steps:

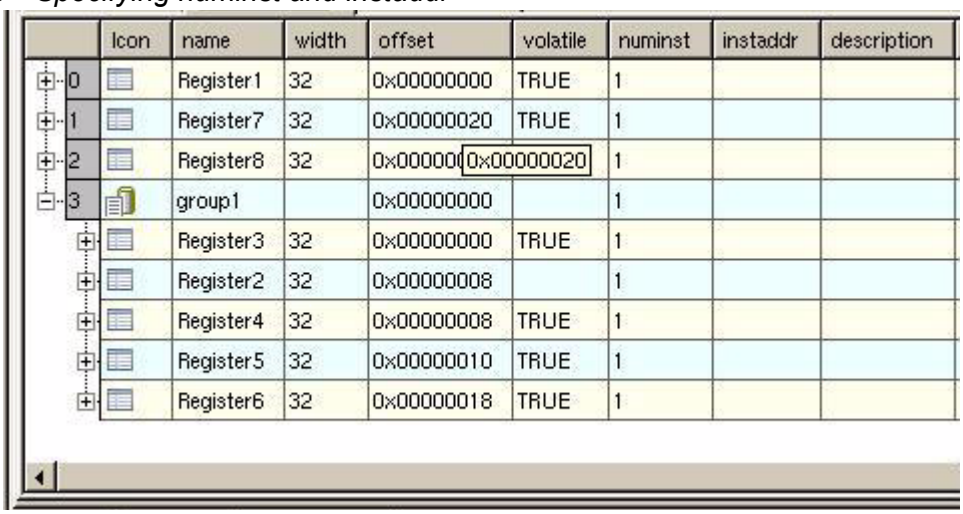
1. Select the rows containing registers that you want to group. You can select multiple rows by:

- Clicking the first row in the range and holding down the SHIFT key while pressing the arrow keys to extend the selection.
- Clicking the first row and holding down the CTRL key while clicking the other desired rows. This method is used to select rows that are not adjacent to each other.
- Clicking the first row in the range and dragging the mouse pointer to the last row in the range.

2. Right-click the selected rows and select the Group menu option from the shortcut menu. This creates a new row with name newGroup and arranges all the rows that were selected for grouping as sub-rows of newGroup.

Now, you can adjust the array_size parameter for the group to specify how many times that group should be replicated in the address map and the array_offset parameter to specify the offset between the last register of one block and the first register of the next block in the replication. Refer to the figure below:

Figure 13-3 Specifying numinst and instaddr



	Icon	name	width	offset	volatile	numinst	instaddr	description
0		Register1	32	0x00000000	TRUE	1		
1		Register7	32	0x00000020	TRUE	1		
2		Register8	32	0x00000 0x00000020		1		
3		group1		0x00000000		1		
		Register3	32	0x00000000	TRUE	1		
		Register2	32	0x00000008		1		
		Register4	32	0x00000008	TRUE	1		
		Register5	32	0x00000010	TRUE	1		
		Register6	32	0x00000018	TRUE	1		

Capturing Component Register Information Using Tcl

To capture the component register information for a component using Tcl, perform ANY of the following actions:

- Create a component by using the new component Tcl command
- Load an existing component by using the load component command.

Refer to *GenSys Tcl Commands Reference Guide* for more details on these Tcl commands.

Then, perform the following steps in the specified order:

1. Add registers (refer to the Adding Registers topic)
2. Add bitfields (refer to the Adding Bitfields topic)
3. Add bit-enums (refer to the Adding Bit-enums topic)

   Note:
   You can also set the visibility of the specified registers or bitfields by using the set_visibility Tcl command. This Tcl command enables you to show or hide the specified registers or bitfields. For more details on this Tcl command, refer to *GenSys Tcl Commands Reference Guide*.

Adding Registers

You can create registers for a component by using the add_register Tcl command. Refer to *GenSys Tcl Commands Reference* for more information.

Note:
The Tcl commands should follow the Tcl syntax. Either give all information in one line or use back-slash for line continuation.

Adding Bitfields

You can add bitfields after defining a register. Creating bitfields is optional. Multiple bitfields may be defined under a single register, as long as LSB-MSB ranges do not overlap.

If no bitfield is defined for a register, no bitfield is assumed to exist for that register.

You can add a bitfield by using the add_bitfield Tcl command. You can also complete the incomplete bitfield information by using the complete_bitfield command. Refer to *GenSys Tcl Commands Reference* for more information.

Note:
The Tcl commands should follow the Tcl syntax. Either give all information in one line or use back-slash for line continuation.

Adding Bit-enums

You can add bit-enums (bit enumeration values) for a bitfield after you have a bitfield. Creating bit-enums are optional. Multiple bit-enums may be defined under a single bitfield, as long as the values are within the range of the bitfield width.

If no bit-enum is defined for a bitfield, no bit-enum is assumed to exist for that bitfield.

You can add a bit-enum by using the add_bitenum Tcl command. Refer to *GenSys Tcl Commands Reference* for more information.

Note:
The Tcl commands should follow the Tcl syntax. Either give all information in one line or use back-slash for line continuation.

Grouping Registers

You can group registers for viewing related groups of registers and also to array a register group. You can create repeated banks of registers, such as a bank of registers for DMA channel 0, a similar bank for DMA channel 1, another similar bank for DMA channel 2, and so on.

The command to define a group is add_group. Refer to *GenSys Tcl Commands Reference* for more information.

Note:
The Tcl commands should follow the Tcl syntax. Either give all information in one line or use back-slash for line continuation.

Capturing Component Memory Information Using GUI

To capture component memory information, perform the following steps:

- Create or modify a component. See the Building Components.
- Click the HW_SW tab in the Data Table window.
- Click the MemoryBlock tab in the DataTable window.

The MemoryBlock tab contains the following columns:

Column	Description
Name	Specifies the name for the memory
Width	Specifies the width of the memory
Offset	Specifies the offset of the memory
size	Specifies the number of addressible locations in the memory block
endianness	Specifies whether big endian or small enndian
Volatile	Specifies whether the memory is volatile or not. Select either TRUE or FALSE from the drop-down list.

Column	Description
Access Type	Specifies the access type for the memory. It can be one of the following values: Execute, Non Accessible, Program Memory, Read, Read Clear, Read Execute, Read Write, Reserved, Sticky, Write, Write Clear.
arraysize	Specifies the array size of the memory block
Bank_Align _Type	Specifies the alignment (PARALLEL, SERIAL, or UNDEF) of the memory block
Description	Enables you to include a description of the memory. This column opens a WYSIWYG editor. The WYSIWYG editor is used primarily to add emphasis or detail to information, which will be passed to datasheets. You can add multiple lines, format texts, add bullets, numbering, and so on in this field. This field also accepts unicode characters.
Short Name	Specifies a short name for the memory.
Comment	Enables you to add comments about the memory.
Visibility	Specifies the visibility condition

To enter values in the MemoryBlock tab, insert a row. To insert rows, refer to the Inserting Rows section.

Now, you can enter the required values in the columns. Use the TAB key to move between fields. You can also use the Edit-> Copy menu option (or CTRL+C) and the Edit-> Paste menu commands (or CTRL+V) to enter information

Refer to the figure shown below:

Figure 13-4 Adding Memory Information

Info | Name | HW_SW | Memory_Map | RemapStates | Por

MemoryBlock | RegisterBlock | Register_Map

	name	offset	width	endianness	access_type
0	memory1	0x00002000	32	SMALL	READ_WRITE
1	memory2	0x00010000	32	SMALL	WRITE

Note that any cell in a GenSys spreadsheet can take a formula. Therefore, while entering sizes for large memories, you can type = followed by an expression or click the formula icon (

) and write the expression in the cell. For example, to enter 17K, type =17*1024.

Generating Component-Level Register Map View

Once you have added component memory information you can review component register maps by using the Register_Map tab in the Data Table window. The component register map view is automatically generated when you click the Register_Map tab. This view also shows the group and alias information to display the complete register map for that component.

Levels of Functions

For embedded APIs that interface to peripherals, there are typically three layers of functions:

Register/Bitfield Access functions: At the lowest level, each bitfield or register should be able to generate a function or macros that provides access to it (optionally using the enums associated with it).

For example, let's imagine there's a UART with two registers. The SW developer should be able to ignore details like offset, LSB, width, and so on, because the tool will insert these later on.

Register: CONTROL	**Write-only**
Bitfield:CONTROL_STATE	Enums:RUN, HALT

Register: CONTROL	Write-only
Bitfield: BAUDEnums:	BAUD_9600, BAUD_4800, BAUD_2400, BAUD_1200
Bitfield: DATABITEnums:	DATABIT_8, DATABIT_7
Bitfield: STOPBITEnums:	STOP_2, STOP_1
Bitfield: PARITYEnums:	ODD_PARITY, EVEN_PARITY, NO_PARITY

Register: STATUS	Read-only
Bitfield: CURRENT_STATE	Enums: HALTED, RUNNING

We should be able to describe accesses to registers/bit-fields using syntax like:

```
REGISTER(CONTROL)
=EXPAND(CONTROL,HALT,BAUD_9600,DATABIT_8,STOP_1,NO_PARITY);
return BITFIELD(STATUS, CURRENT_STATE);
```

Register / Bitfield / Enum Names

The goals of names should be to minimize repetitive typing with minimal possibilities for errors. Matches will first be tried against ‘short names’ in the schema – these should consist of word characters (a-z, A-Z, _, 0-9) and start with a non-digit. If no short names match, then field-names and register-names will be used. If the match found is non-unique (i.e. more than one element matches at either the bitfield or enum level, then an error should be reported).

The period (‘.’) will be used as a separator between register groups, registers, bitfields, and other elements. The square braces (‘[’ and ‘]’) will be used as an indicate for instance elements. For example, DMA[8] would indicate the 9th instances of a DMA array element.

Register names will generally be unique. If a register is not unique for a peripheral, its register group must be specified as part of its name. For example, DMA[8].START_ADDR indicates the START_ADDR register in the 9th instance of the DMA register group.

Bitfield names may or may not be unique. If not, then the register (and, if necessary, register group) must be specified as part of its name.

Once the bitfield/register has been identified, then enums (if used) can be matched. If the match occurs at the bitfield level, then the bitfield-value’s enumName must be a valid match

for either that bitfield, or the register it's in, or for that component. (enum names should be checked for in that order – the 'most local' enum has precedence).

Write-To-Decrement Registers

The EMAC has a set of statistics that record events associated with frame traffic (for example, how many good transmit frames sent, how many jabber frames received, and so on). The statistics values are cleared to zero, 38 clocks after the rising edge of reset. When the MIIEN bit in the MACCONTROL register is set, all statistics registers are write-to-decrement. The value written is subtracted from the register value with the result stored in the register. If a value greater than the statistics value is written, then zero is written to the register (writing FFFF FFFFh clears a statistics location). When the MIIEN bit is cleared, all statistics registers are read/write (normal write direct, so writing 0000 0000h clears a statistics location). All write accesses must be 32-bit accesses.

The routine for getting the statistics is as shown:

```
static void emacUpdateStats( EMAC_Device *pd ){    int             i;
volatile Uint32 *pRegAddr;    Uint32         *pStatAddr;    Uint32
statval;    pRegAddr = EMAC_ADDR(RXGOODFRAMES);    pStatAddr = (Uint32
*)(&pd->Stats);    for( i=0; i<EMAC_NUMSTATS; i++ )    {        statval =
*pRegAddr;       *pRegAddr++ = statval;       statval += *pStatAddr;
*pStatAddr++ = statval;    }}
```

(It would be handy to have an easy way to get the number of registers in a group – either a constant or function to call that would give the equivalent of "EMAC_NUMSTATS") in the function above.)

Creating Channels and Bridges

GenSys enables you to model bus IPs by using channels and bridges. Channels and bridges can be added only for bus type components. To create a bus type of components, specify the value of the -componentType argument of the new component Tcl command as BUSTYPE.

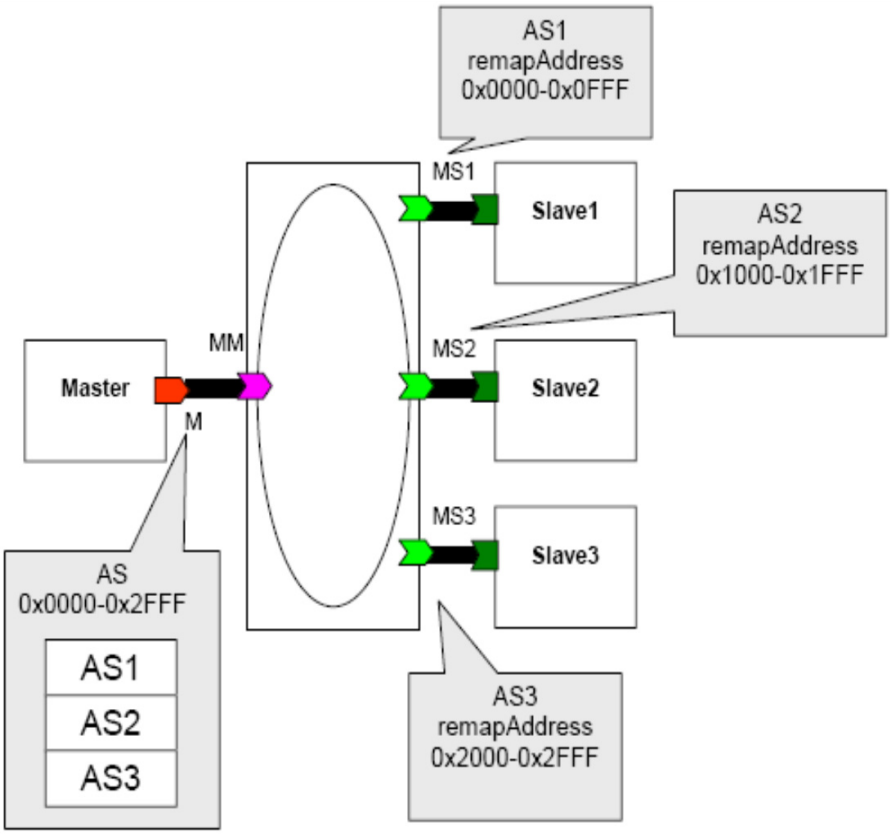
Creating Channels

A channel refers to a structure that encapsulates all the component-level internal connections between mirrored masters and slave interfaces in a bus IP. It represents a simple wiring interconnect or a more complex structure such as a bus.

Consider a scenario in which you have a master component, M, and three slave components, slave1, slave2, and slave3. Consider that these components are connected through a channel through mirrored master interface, MM, mirrored slave interfaces, MS1,

MS2, and MS3. Here the master, M, is connected to three slave memories each of size 4Kb using the channel. This entire scenario is illustrated in the following figure:

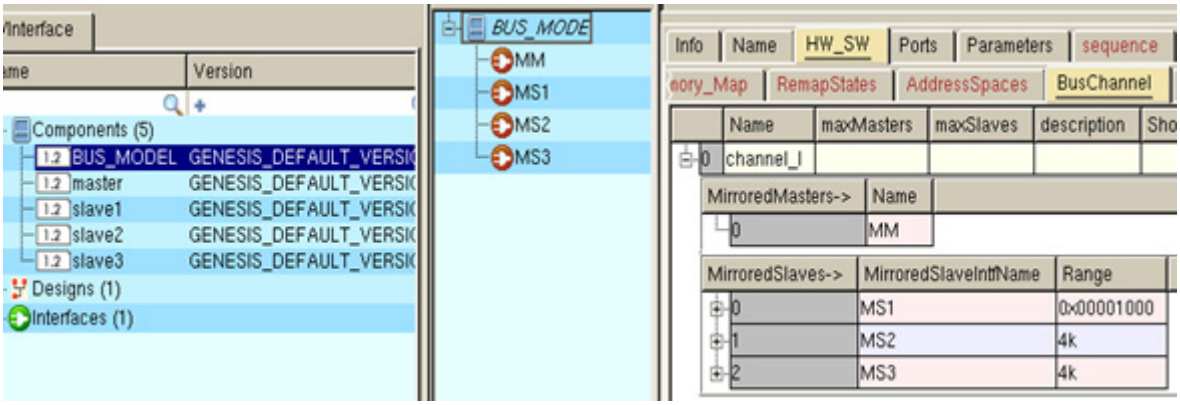
Figure 13-5 Diagrammatic Illustration



In a channel, a memory-map is defined on each mirrored slave by using the base address element. This element refers to the base address of the mirrored slave as seen through this channel and the range of the addressable memory of mirrored slave.

You can add a channel in the HW_SW.BusChannel table for a bus type component, as shown in the following figure:

Figure 13-6 Adding Channel in the BusChannel Table



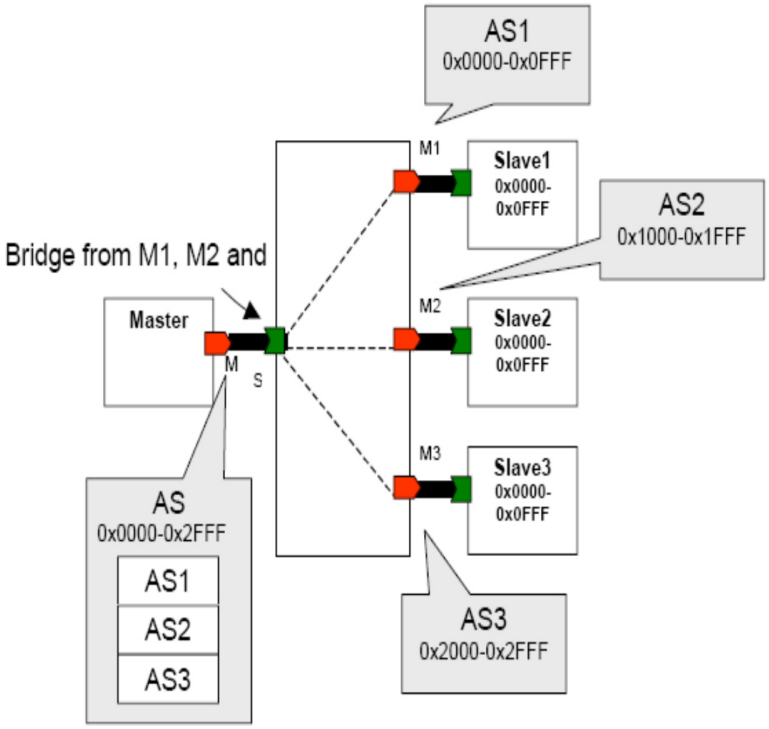
You can also add a channel by using the add_channel Tcl command. For more details on this Tcl command, refer to the *Tcl Commands Reference Guide*.

Creating Bridges

Bridge is another mechanism, besides channel, for describing the internal connections for a bus. For each slave interface on a bus component, you can define a bridge path to the master interfaces.

For example, consider the following figure:

Figure 13-7 Creating Bridges



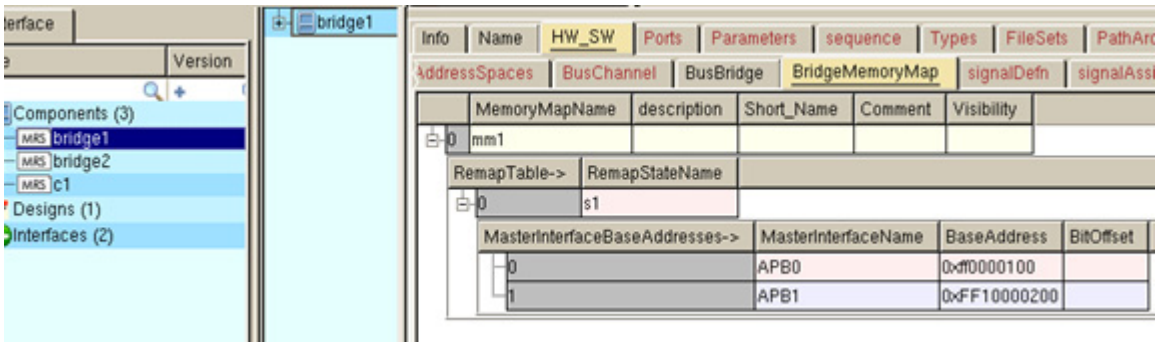
Here, there is a bridge path from the slave interface, S, to the master interfaces, M1, M2, and M3. In addition, an address space is defined on each master interface.

Bridges are of two types, namely transparent bridges and opaque bridges. In a transparent bridge, there is no memory-map section. In this case, the address space on each master interface is static.

In an opaque bridge, address space of each master interface is not static. However, the range is fixed in this case. In addition, there is a memory-map reference on each slave interface. Under this memory map section, you can define the starting base address of the master interface.

You can define a memory-map for a bridge in the HW_SW.BridgeMemoryMap table, as shown in the following figure:

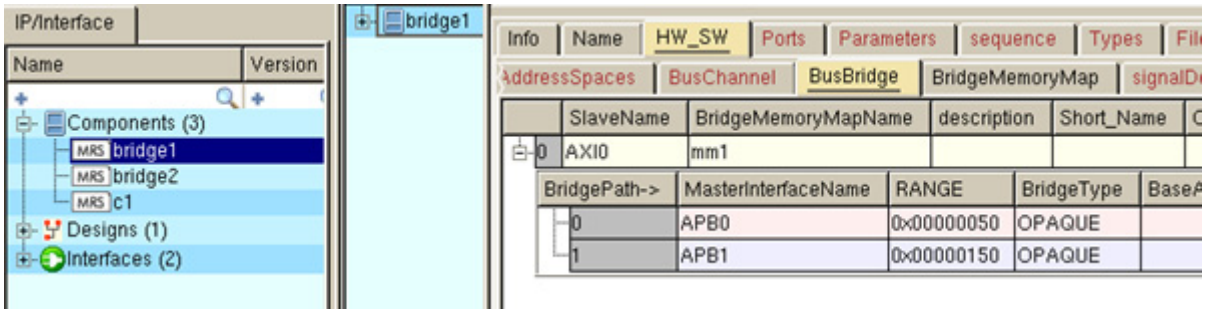
Figure 13-8 Define a Memory-Map for a Bridge in the BridgeMemoryMap Table



Alternatively, you can define a bridge memory-map by using the add_bridge_memory_map Tcl command. For more details on this Tcl command, refer to the *Tcl Commands Reference Guide*.

You can create a bridge in the HW_SW.BusBridge table, as shown in the following figure:

Figure 13-9 Creating a Bridge in the BusBridge Table



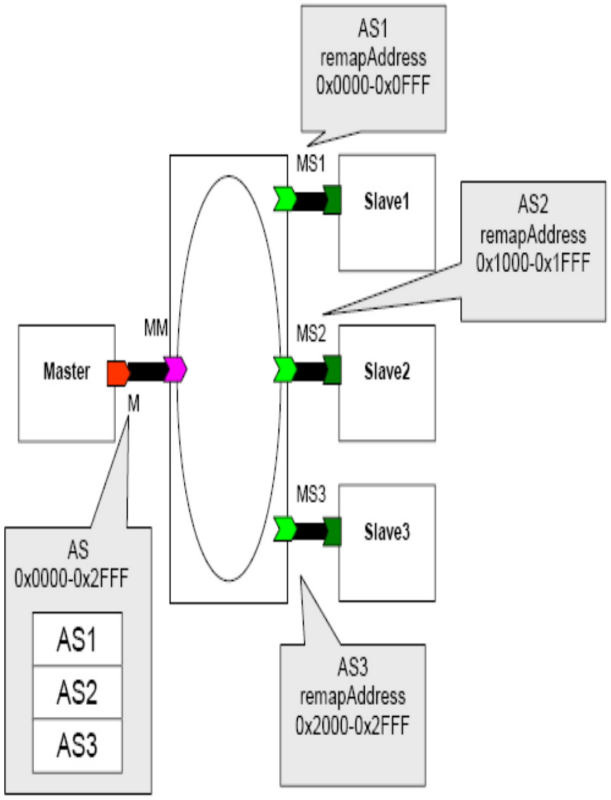
Alternatively, you can define a bridge by using the add_bridge Tcl command. For more details on this Tcl command, refer to the *Tcl Commands Reference Guide*.

Understanding Elaboration

During elaboration, GenSys creates a virtual memory-map between the actual requestor and target present in the path. For channels, this virtual memory map is created dynamically between each pair of master and slave components connected through mirrored master and mirrored slave. For bridges, this virtual memory-map is created dynamically between each pair of master and slave components connected through each bridge path.

For example, consider the following figure:

Figure 13-10 Elaboration



Here, GenSys creates a memory-map map view between non-bus masters and non-bus slaves by using the information present in a channel/bridge.

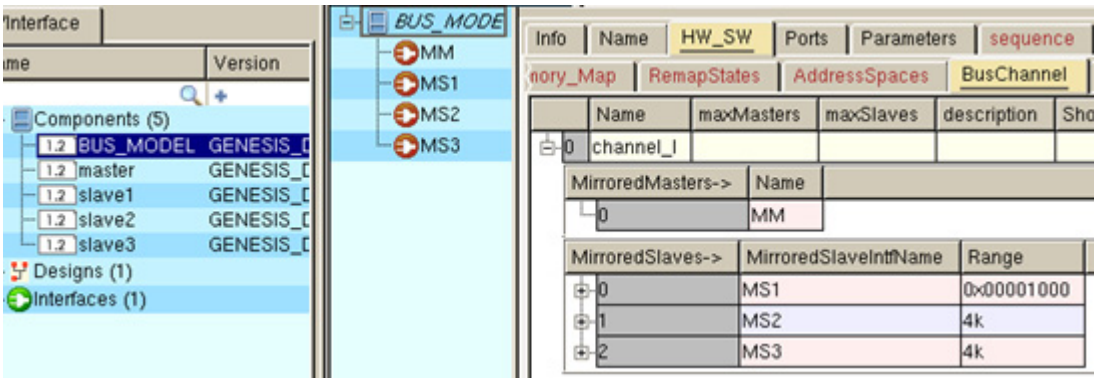
Example of Elaboration in Case of Multiple Channels and Bridges

This section provides you an example of elaboration in case of multiple channels and bridges. This is discussed in the following steps:

1. Creating a Channel

This step creates a channel, Channel_I, in the HW_SW.BusChannel table, as shown in the following figure:

Figure 13-11 Creating a Channel



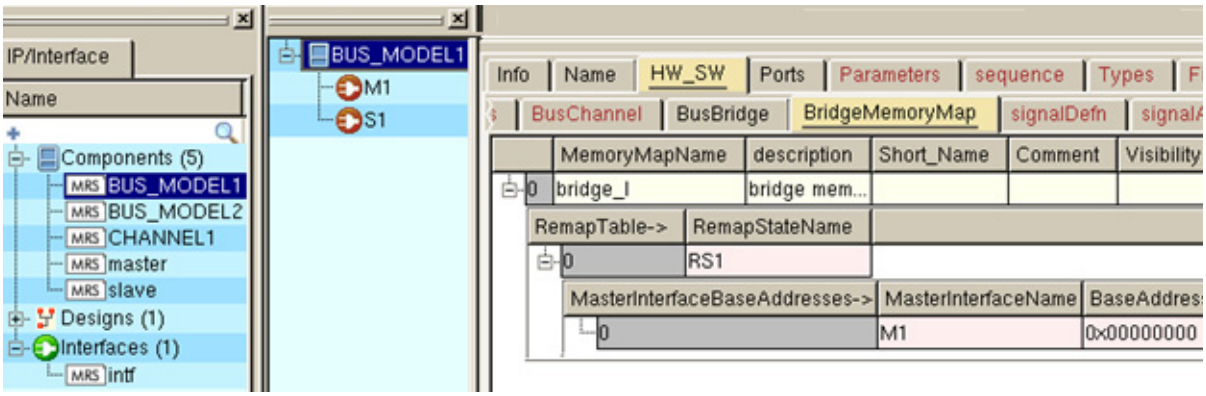
2. Creating Bridges

This step create bridges and bridge memory maps by using two BUSTYPE components, BUS_MODEL1 and BUS_MODEL2. This is discussed in the following sub-steps:

2.1 Creating Bridge Memory Map for BUS_MODEL1

The bridge memory map details for BUS_MODEL1 is specified in the HW_SW.BridgeMemoryMap table, as shown in the following figure:

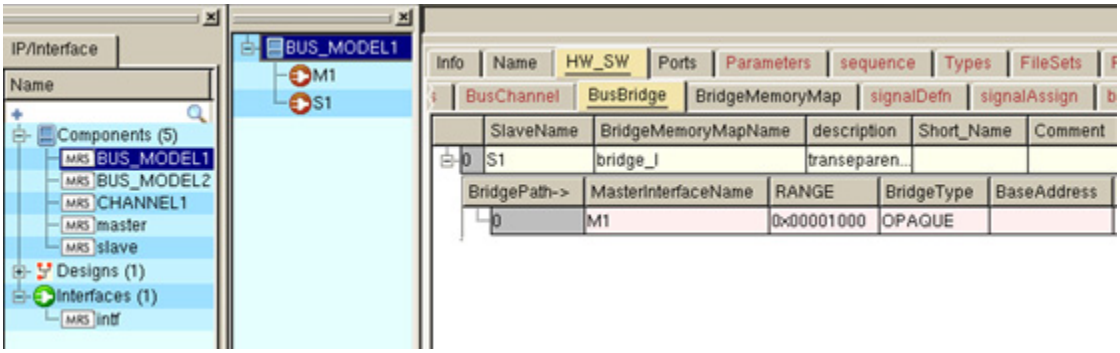
Figure 13-12 Creating Bridge Memory Map



2.2 Creating Bridge Path for BUS_MODEL1

The details of the bridge path defined for the slave interface, S1, on the bus component, BUS_MODEL1, is specified in the HW_SW.BusBridge table, as shown in the following figure:

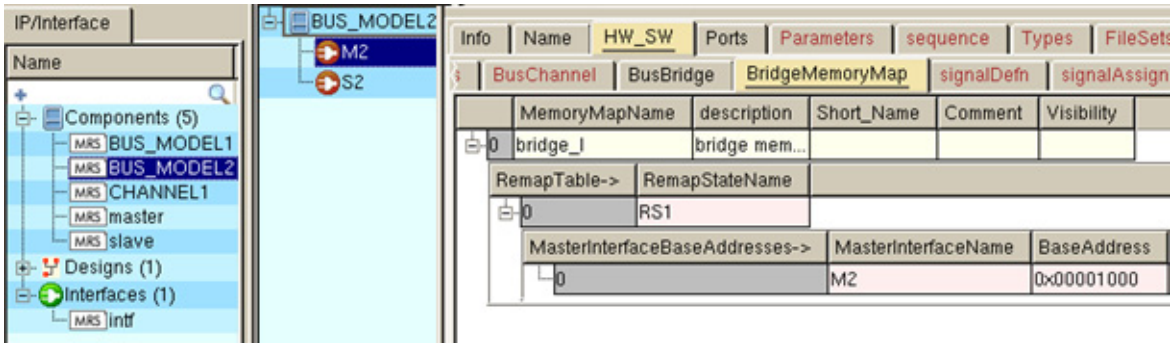
Figure 13-13 Creating a Bridge Path



2.3 Creating Bridge Memory Map for BUS_MODEL2

The bridge memory map details for BUS_MODEL2 is specified in the HW_SW.BridgeMemoryMap table, as shown in the following figure:

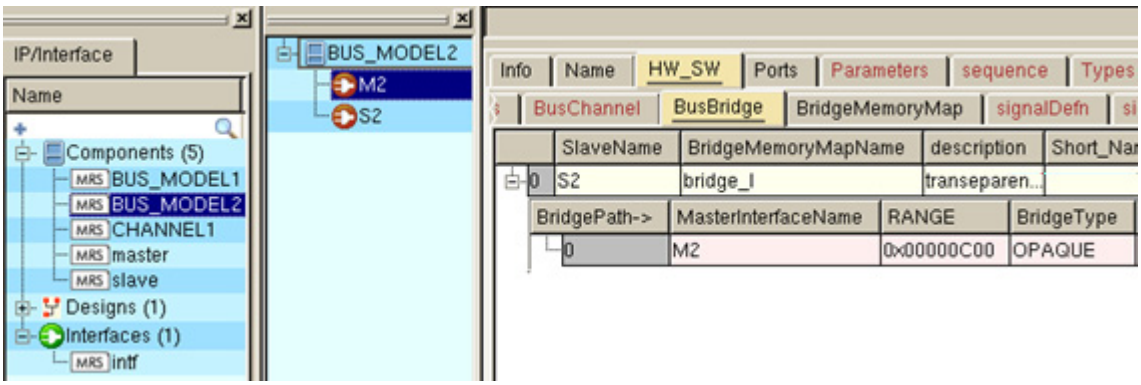
Figure 13-14 Creating Bridge Memory Map



2.4 Creating Bridge Path for BUS_MODEL2

The details of the bridge path defined for the slave interface, S2, on the bus component, BUS_MODEL2, is specified in the HW_SW.BusBridge table, as shown in the following figure:

Figure 13-15 Creating Bridge Path

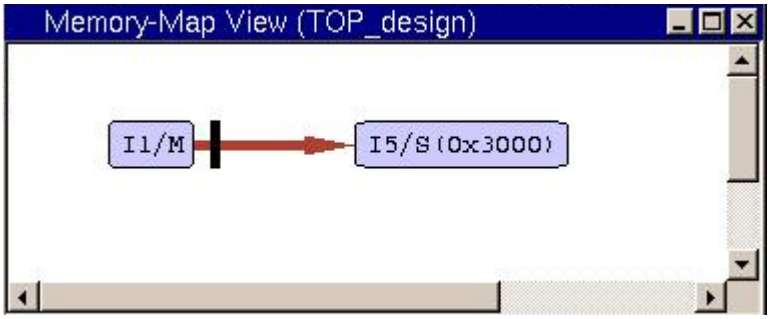


3. Generating Memory Map

To generate a memory map, select the GenerateMemoryMap option from the Generator menu. Once you generate the memory map, you can view the memory map by clicking the Memory map Viewer button (

). When you select this button, GenSys displays the Memory-Map View window, as shown in the following figure:

Figure 13-16 Memory-Map View Window



Handling Remap States During Elaboration

The component may contain a list of remap state for which the base addresses of the target can be remapped to the values associated with different states.

In case no remap state is provided, elaboration logic will proceed by using the default base addresses associated with the targets. If a list of remap states are available with the component, take the data corresponding to the first state (first entry present in the list) and elaborate accordingly (that is, create memory-map).

Note:

Currently, GenSys does not support state-specific maps specified by the user. However, GenSys support maps that are not state-specific.

14

Working with Memory Maps

Memory maps establish a relationship between one or more bus interfaces, called entry points, and:

- Registers in a leaf-level component, that is, at what address is each register seen from the entry points
- Bus interfaces, that is, at what address is each bus interface seen from the given entry points

You can define memory maps for designs and components. GenSys provide various tables in the data table window for designs and components in which you can define and generate memory maps. In addition, you can also view the graphical representation of memory maps and registers by using the Register/Memory-Map Builder dialog.

This chapter covers the following topics:

- Understanding Memory Maps
- Generating Memory Map View
- Viewing Graphical Representation of Register and Memory-Maps

Understanding Memory Maps

Memory maps can be at a design-level or a component-level.

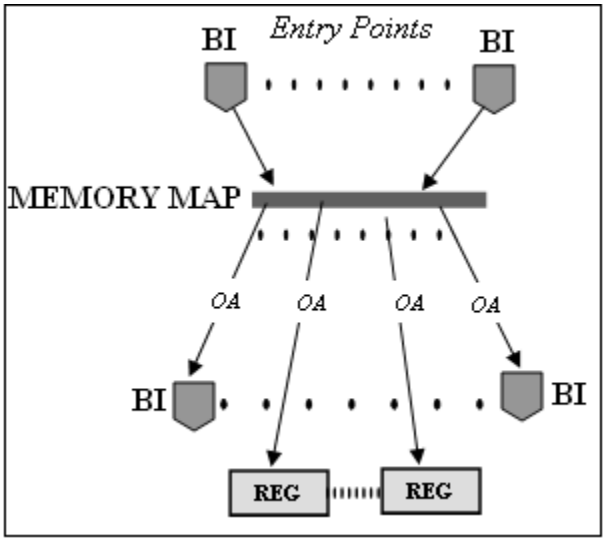
Design-level memory maps establish a relationship between bus interfaces and instance's bus interfaces or bus interfaces of the current design. Component-level memory maps establish a relationship between bus interfaces and registers or other bus interfaces. A component can contain more than one memory map.

You can define memory maps at each level of a hierarchy, where each level is seen as a black box from the level above.

You can hierarchically specify memory maps for each sub-system in a design and at the top design level. While generating a top-level design memory map view, hierarchical sub-system level memory maps are elaborated to generate a flat memory map view of the entire design. You can specify address map information by using GUI or Tcl commands.

The following figure displays a typical case of a memory map:

Figure 14-1 A Typical Case of a Memory Map



In the above figure, OA refers to the offset address (start address) of the target element. For register target elements, this value is optional. If you do not specify the offset address for register target element, GenSys uses the register offset value by default.

You can specify the end address of the target element by using the -end argument of the add_target_map Tcl command. If you do not specify this argument, GenSys uses the size of the register or target interface (size of the interface is defined using address_range field) to calculate the end address (end address = start + size-1).

Specifying MAU for a Requestor Interface

You can specify MAU value (in bytes) for each requestor interface in the access_size column under the Info tab of that interface.

If you do not specify any value in the access_size column, then by default, the value is picked from the register_width column under the Info tab of a component. This value is defined in bits for the component, but it is converted into bytes while calculating MAU for the requestor interface. If no value is defined in the register_width column of the component, 1 byte is considered as the default register width in the register_width column for that component.

All start addresses entered in memory maps should be aligned to the requestor MAU. If not they are rounded up to the nearest requestor MAU, a warning is reported.

For example, if MAU is defined as 4 and you have defined the start address for an element as 7, the address is rounded off to 8 and a warning is reported.

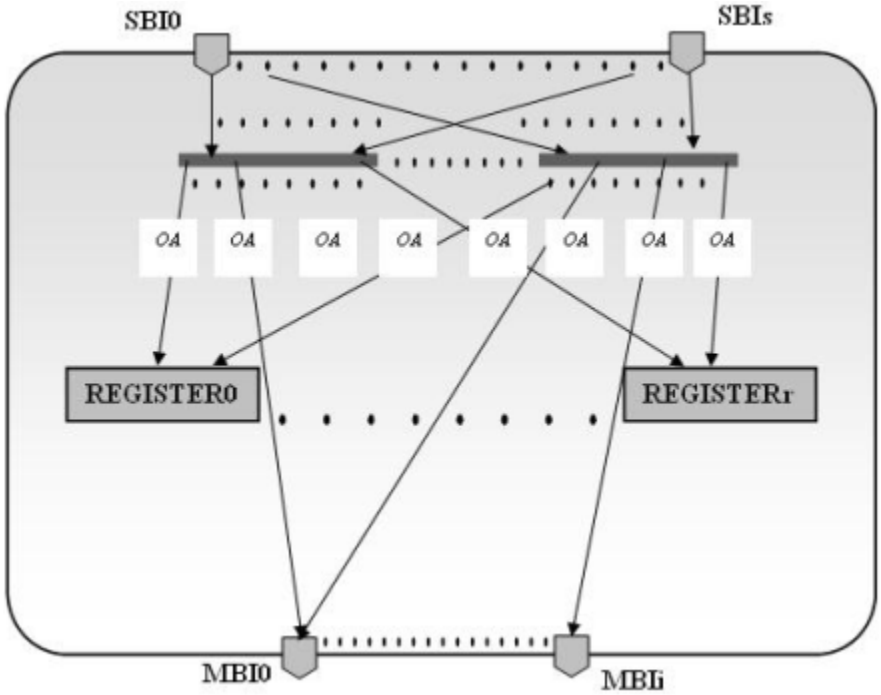
You should specify an end addresses as either <*start-address*> + <*register-width*>/8 - 1 or (<*a-higher-multiple-of MAU*>) - 1.

For example, if MAU is defined as 4 and the width of a register is 16 bits with the start address as 0, the end address can be 0x1 or 0x3, 0x7.

Component-Level Memory Maps

Component-level memory maps define a relationship between slave bus interfaces and registers or master bus interfaces. The following figure illustrates a component-level memory map:

Figure 14-2 Component-Level Memory Map



You can specify component-level memory map information in the Memory_Map_Table table under the HW_SW.Memory_Map tab for a component.

The Memory_Map_Table table contains the following columns:

Column	Description
Name	Specifies the name of the memory map
description	Specifies a description for each memory map. This column opens a WYSIWYG editor. The WYSIWYG editor is used primarily to add emphasis or detail to information, which will be passed to datasheets. You can add multiple lines, format texts, add bullets, numbering, and so on in this field.
Short_Name	Defines a unique name for the particular memory map.
Comment	Specifies any comments for the memory map.
Visibility	Shows visibility of an memory map.

The Memory_Map_Table table further contains two sub-tables, Requestors and Targets, which enables you to specify the details for requestors and targets, respectively.

The Requestors table contains the following columns:

Column	Description
Requestor_inst	Specifies name of requestor instance for the current memory map.
Requestor_intf	Specifies the interface name on the requestor instance for which memory map is defined.
description	Specifies a description for each requestor. This column opens a WYSIWYG editor. The WYSIWYG editor is used primarily to add emphasis or detail to information, which will be passed to datasheets. You can add multiple lines, format texts, add bullets, numbering, and so on in this field.
Short_Name	Defines a unique name for the requestor.
Comment	Specifies any comments for the requestor.
Visibility	Shows visibility of the requestor.

The Targets table contains the following columns:

Column	Description
Start	Specifies the start address information for the target
End	Specifies the corresponding end address for the target.
Feedthru_Offset	Specifies the feedthrough offset information for the target
Target_inst	Specifies name of target instance for current memory map.
Target_intf	Specifies the interface/register/memory name on the target instance for which memory map is defined.
Element_Type	Specifies the element type (INTERFACE, MEMORY, REGISTER, REESRVED) for the target

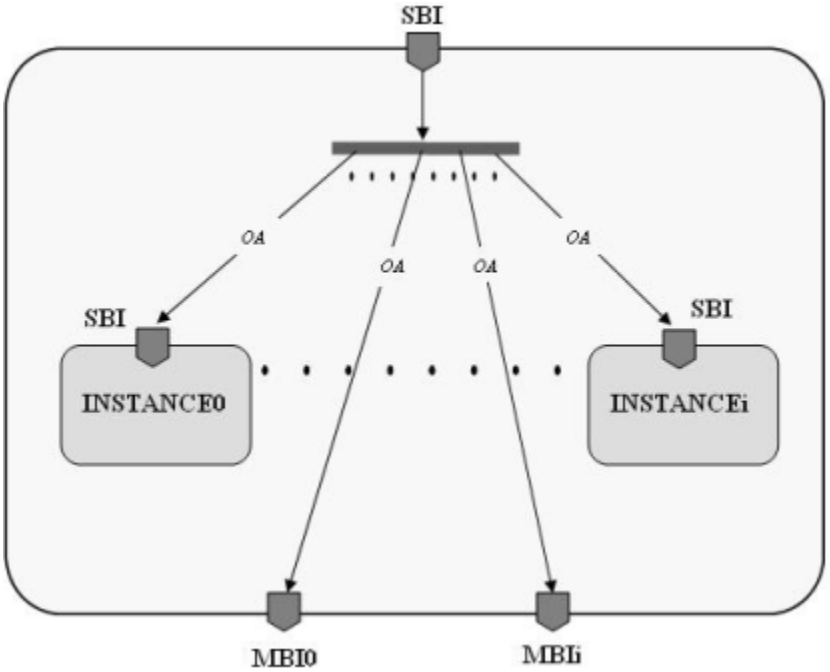
Column	Description
description	Specifies a description for the target. This column opens a WYSIWYG editor. The WYSIWYG editor is used primarily to add emphasis or detail to information, which will be passed to datasheets. You can add multiple lines, format texts, add bullets, numbering, and so on in this field.
Short_Name	Defines a unique name for the target.
Comment	Specifies any comments for the target.
Visibility	Shows visibility of the target.

Design-Level Memory Map

In a design-level memory map, the requestor interface (that is, SLAVE bus interface on the design or MASTER bus interface on the instance) can request SLAVE bus interface of other instances or MASTER bus interface of the design. If the design contains sub-system instances, you can define hierarchical memory maps. On elaboration of such hierarchical memory maps, flat-level memory map view is generated for the design.

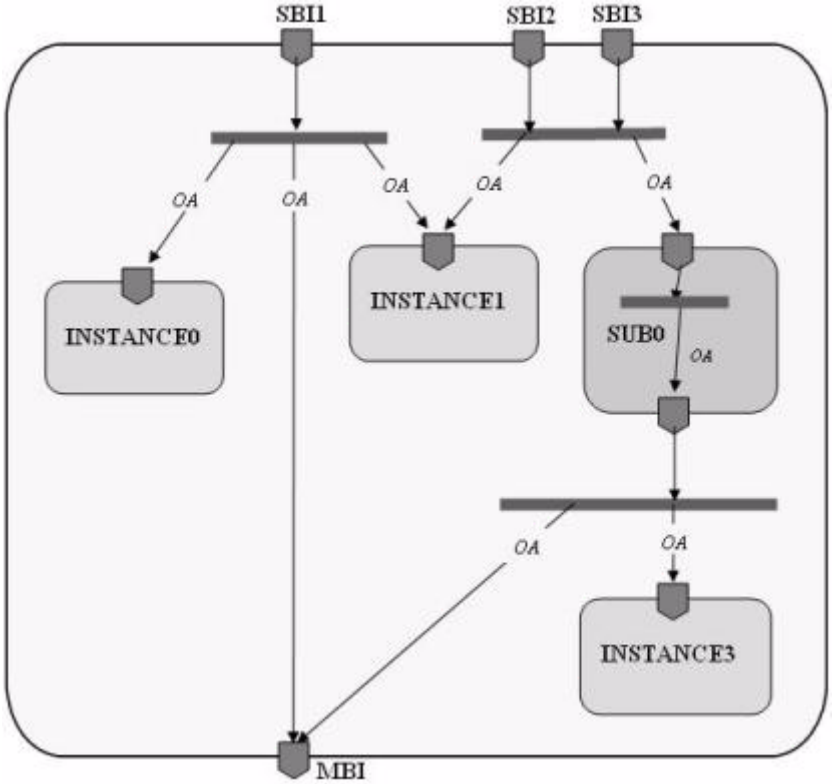
Following figure illustrates a simple design-level memory map:

Figure 14-3 A Simple Design-Level Memory Map



In a more complex case of design-level memory maps, feedthrough memory mapping can be defined if the design contains sub-system instances, as shown in the following figure:

Figure 14-4 A Complex Case of Design-Level Memory Map



In the above figure, feedthrough memory mapping is defined in the sub-system, SUB0.

You can define design-level memory map information in the HW_SW.Memory_Map.Memory_Map_Table table for a design. The Memory_Map tab for a design is similar to the Memory_Map tab for a component.Like a component, you can specify the memory map details for a design in the HW_SW.Memory_Map tab for a design. This tab also contains two sub-tables, Requestors and Targets, which enable you to specify the requestors and target information for the design-level memory map.

Specifying Memory Type

You can specify the type of memory address entered in the Start and End columns of the Memory_Map_Table table. The allowable types are shown in the following table:

Decimal Integer	C-Style Hexadecimal	C-Style Binary
Verilog Hexadecimal	Verilog Binary	VHDL Hexadecimal

VHDL Binary

Note:

If you enter any data in these columns and select the type as C-Style Hexadecimal, all the subsequent data entered will be displayed and read as Hexadecimal type only.

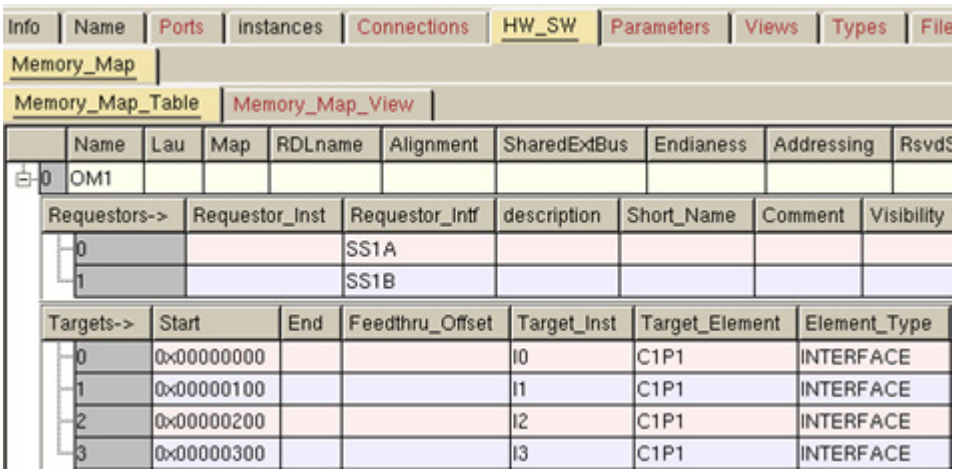
To select the type, right-click on the column header of Start, End or Size columns and select the Views menu option from the shortcut menu. This displays the all the allowed types for these columns.

Adding, Modifying, and Viewing Memory-Maps

You can add or modify memory map information in the Tabular or Canvas view.

Adding Memory Maps in the Tabular View

You can specify memory-map information in the HW_SW.Memory_Map.Memory_Map_Table table, as shown in the following figure:



In the above table, specify memory-map details, such as requestors and targets.

Adding Memory Maps in the Canvas View

To add a memory-map in the canvas view, perform the following steps:

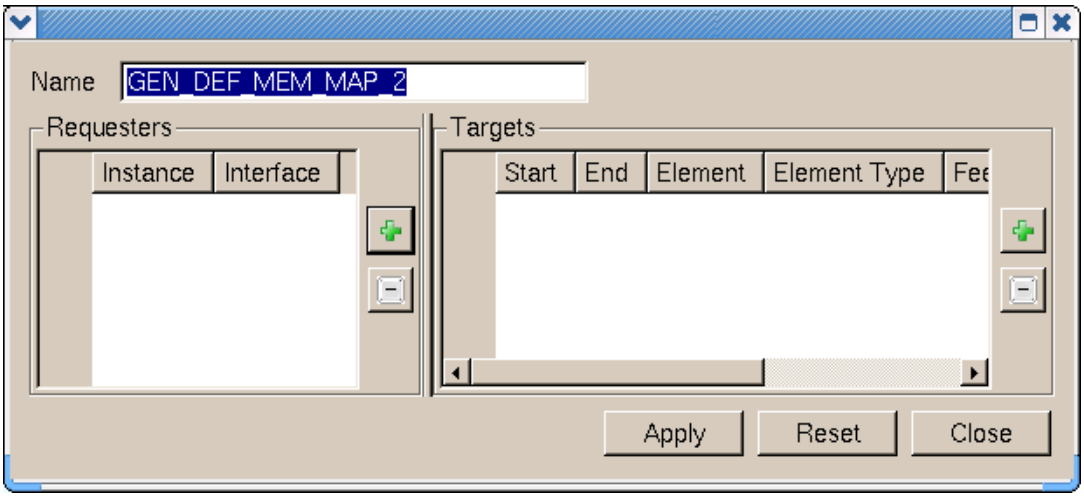
1. Select the Registers View option from the Schematic Mode drop-down list in the toolbar.
2. Click the

3. button to enter in the edit mode.
4. Click the

5. button to add a memory map.

   Alternatively, right-click in the canvas view and select the Add Memory Map option from the shortcut menu.

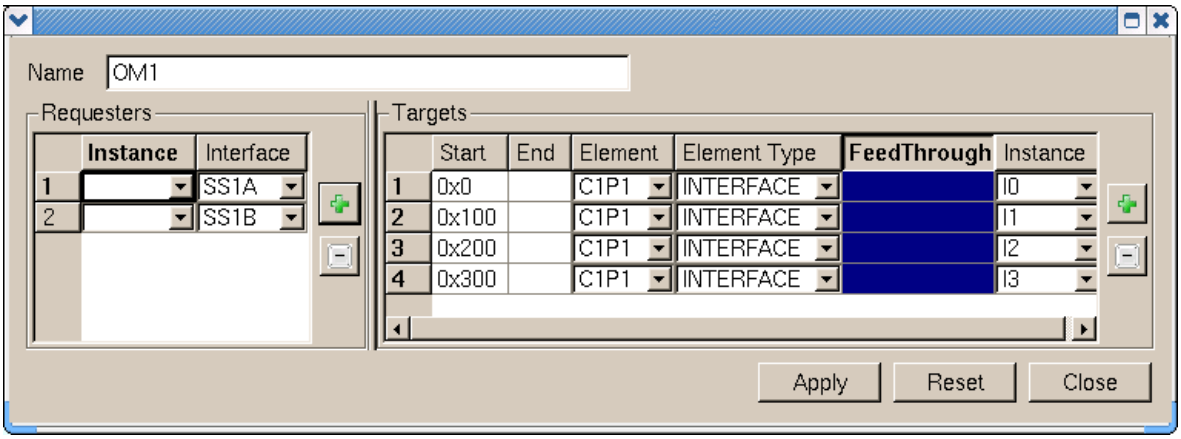
   The following dialog appears:



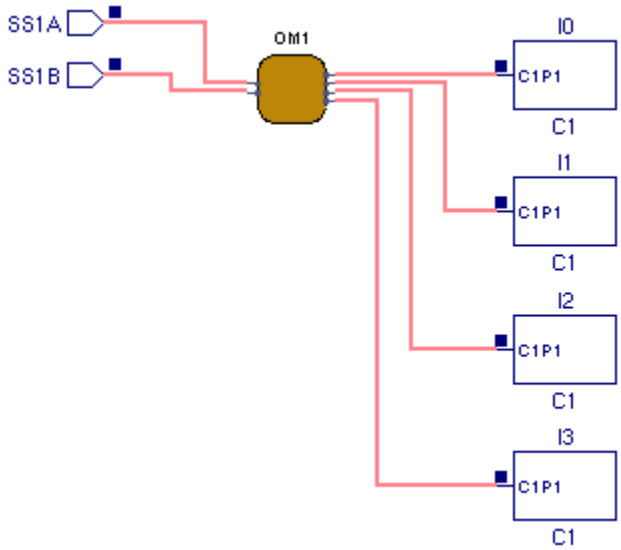
6. In the above dialog, specify requestors and targets.
7. Click the Apply button to generate the memory map with the specified requestors and targets.

After performing any of the above steps, GenSys creates a new memory-map (with a default name) for the design/component.

For example, consider the following dialog in which you have specified the requestor and target details:



Based on the details specified in the above dialog, the following memory map appears in the canvas view:



Modifying Memory Maps

You can modify details of a memory-map in any of the following ways:

- Double-click on a memory-map under the Design Browser or in the canvas view.

This displays the memory-map editor dialog which contains the details of requestors and targets of the selected memory-map. Modify information in this dialog.

- Select the HW_SW.Memory_Map.Memory_Map_Table table and update information of a memory-map.

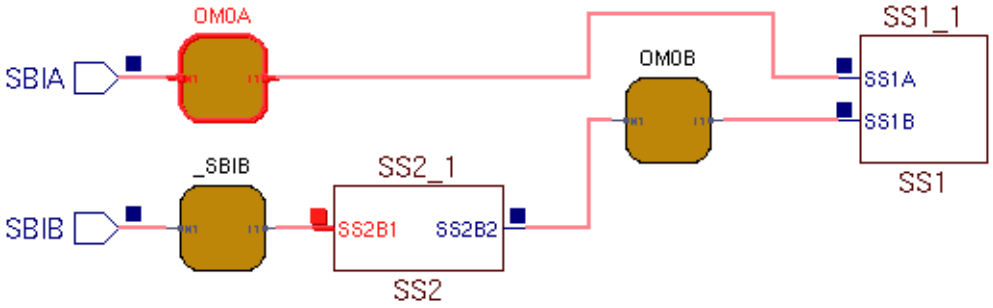
Adding Requestors and Targets in a Memory Map in the Canvas View

To add a requestor or a target in a memory-map in the canvas view, perform the following steps:

1. Select the Registers View option from the Schematic Mode drop-down list in the toolbar.
2. Click the

3. button to enter in the edit mode.
4. Select a memory-map and an interface keeping the <Shift> keyboard key pressed.

   The following figure shows the selected objects (OMOA memory-map and SS2B1 interface):



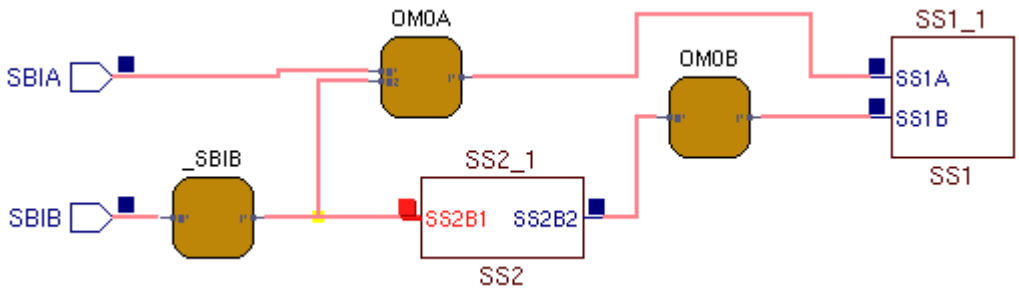
5. Right-click on the selected objects, and select the Add as requestor or Add as target option from the shortcut menu.

   Alternatively, click the

   button or the

button from the canvas toolbar.

After performing the above steps, a requestor or target appears in the canvas view. For example, in the above figure, if you add a requestor for the selected objects, GenSys adds a requestor and the Canvas View changes to the following:



Viewing Memory Maps with Assembly

Apart from the Assembly view, which is the default view, and the Register view, there is third view available. This view is called Assembly + Registers. This view enables you to see the Registers View superimposed on the Assembly View. In this view, you cannot make any edits.

To access this view, select the Assembly + Registers option from the Schematic Mode drop-down list in the toolbar.

Generating Memory Map View

To generate a memory map view, select the GenerateMemoryMap option from the Generator menu. Alternatively, use the gen_memory_map Tcl command to generate a memory map view.

GenSys generates the memory map view in the Memory_Map_View table under the HW_SW.Memory_Map tab of the design/component. This table displays a flattened memory map view in which it displays the offset, requestor instance interface, and target instance interface/register information to show the complete memory map for the design/component. Design-level memory map view displays up to instance interface-level mappings. Component memory map view also displays the register-level mappings.

To ensure that the address mapping details are correct, GenSys validates the address range specified in the Memory_Map_Table table. GenSys validates these entries for each element being requested or addressed by extracting the details of the registers in the master component. Then, GenSys uses these details to calculate the minimum range that must be

accessed by the requester, depending on the requester MAU, after aligning their offsets to MAU. Based on the start address specified by the user and the minimum required range, the minimum required value for the end address is calculated. Considering the calculated end value, GenSys validates the end value specified by the user in the following manner:

- If the end address is not specified, GenSys uses the calculated end value.
- If the calculated end address is not equal to the end address specified by the user, GenSys flags a warning. In addition, the end address specified by the user is replaced with the computed value.
- If the requested address range overlaps for a two or more target elements, GenSys flags a warning message.
- Address ranges which are not requested by that requestor are marked as RESERVED (in pink color)

GenSys creates a separate column for each requestor-instance/requestor-interface combination in the Memory_Map_View table. The reserved address range in each column is highlighted in pink color.

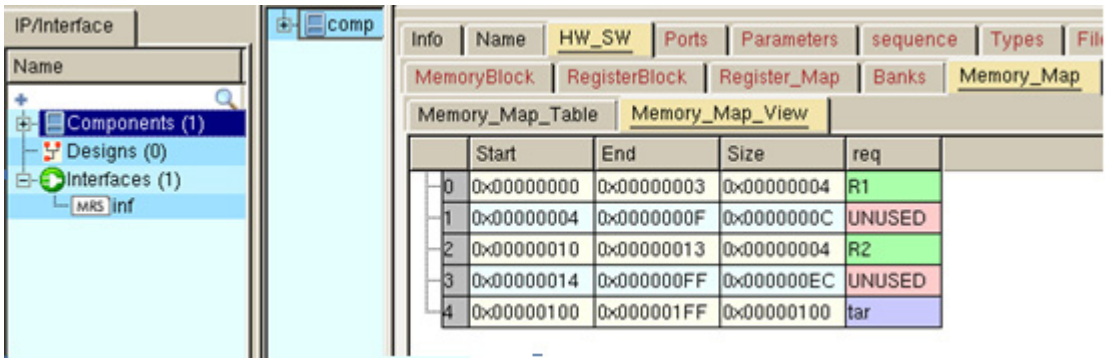
Generating Component-Level Memory Map View

Consider an example in which you create a component-level memory map by using the following Tcl commands:

```
new interface infsave new component compadd_interface -name req
-interface_ref inf -type SLAVEadd_interface -name tar -interface_ref inf
-address_range 0x100
add_register -name R1 -offset 0x00 -width 32add_register -name R2 -offset
0x10 -width 32
#add memory mapadd_memory_map -name map1add_requestor_map -requestor_intf
reqadd_target_map -register R1add_target_map -register R2add_target_map
-target_intf tar -start 0x100save comp#elaborate memory map info and
generate memory map viewgen_memory_map
```

When you generate the memory map view for the above memory map, GenSys populates the Memory_Map_View table, as shown in the following figure:

Figure 14-5 The Memory_Map_View Table



Here, the unused address range in each column is highlighted in pink color. In addition, the register and bus interface are shown in different colors.

GenSys provides you a simpler way to add registers and memory map data for a component, and simultaneously view the register bitfield diagram and flat memory map view in a much more interactive way. You can do this in the Register/Memory-Map Builder dialog. For more details on this dialog, refer to the Viewing Graphical Representation of Register and Memory-Maps section.

Generating Design-Level Memory Map View

A design may or may not contain subsystems. Memory mapping will be different in each of these case. Let us discuss each case in detail with examples.

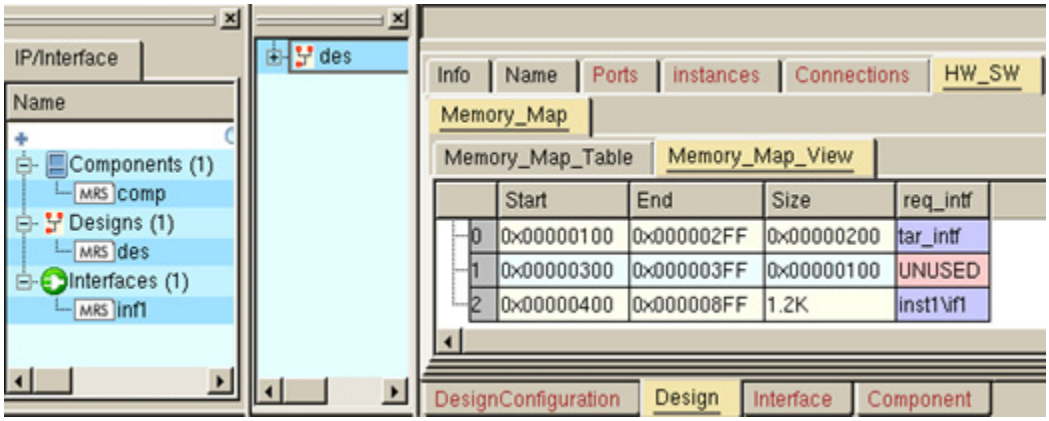
Case1: Design Does Not Contain Any Subsystem (Simple Case)

Consider the following example in which the design does not have any subsystems, and memory mapping is defined for each leaf-level instances inside design:

```
new interface inf1savenew component compadd_interface -name if1
-interface_ref inf1 -type SLAVE-address_range 0x100
new design desadd_instance -name inst1 -master compadd_interface -name
req_intf -interface_ref inf1 -type SLAVE
add_interface -name tar_intf -interface_ref inf1 -type MASTER
-address_range 0x200
#add memory map infoadd_memory_map -name map1add_requestor_map
-requestor_intf req_intfadd_target_map -start 0x100 -target_intf
tar_intfadd_target_map -start 0x400 -end 0x8ff -target_instinst1
-target_intf if1save#elaborate memory map info and generate memory map
viewgen_memory_map
```

The elaborated memory map view for the above memory map is shown in the following figure:

Figure 14-6 An Elaborated Memory Map View



Here, the unused address range is highlighted in pink color.

Case2: Design Contains subsystems (Complex Case)

Now let us consider a design that has subsystem instances having feedthrough and window mapping. Consider that feedthrough offset (0x100) is specified for the SS2_1 subsystem memory map. Also, explicit end address (0x3ff) is specified for feedthrough mapping in subsystem SS2_1 memory map. All the target nodes, which falls outside the given address range window, would be ignored.

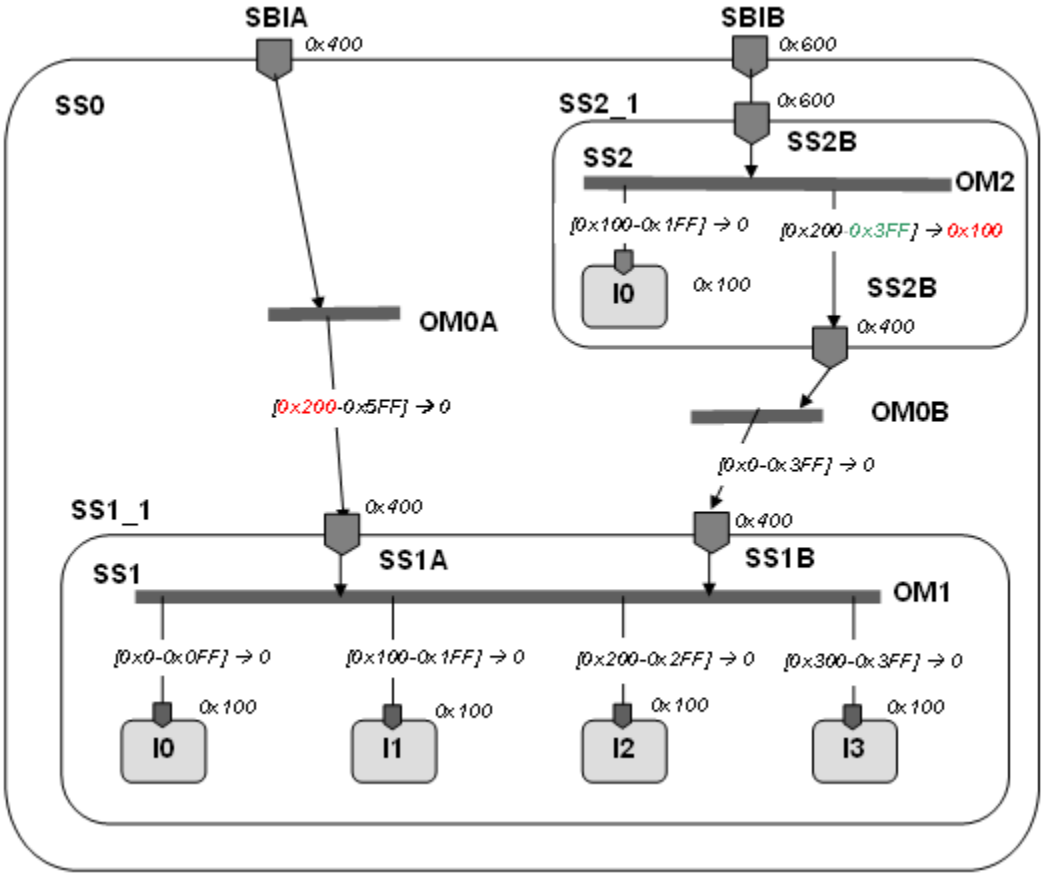
The following example creates a memory map for a complex design:

```
# make the dummy black box component used as leaf cell# "C1"
new interface D_Isave D_Inew component C1
add_interface -name C1P1 -interface_ref D_I -address_range 0x100 -type
SLAVE
save C1
#make sub system SS1
new design SS1add_instance -name I0 -master C1add_instance -name I1
-master C1add_instance -name I2 -master C1add_instance -name I3 -master
C1add_interface -name SS1A  -interface_ref D_I -address_range 0x400 -type
SLAVEadd_interface -name SS1B  -interface_ref D_I -address_range 0x400
-type SLAVE
add_memory_map -name OM1add_requestor_map  -memory_map OM1
-requestor_intf SS1Aadd_requestor_map  -memory_map OM1 -requestor_intf
SS1Badd_target_map -memory_map OM1 -target_inst I0 -target_intf C1P1
-start 0x0add_target_map -memory_map OM1 -target_inst I1 -target_intf
C1P1 -start 0x100add_target_map -memory_map OM1 -target_inst I2
target_intf C1P1 -start 0x200add_target_map -memory_map OM1 -target_inst
I3 -target_intf C1P1 -start 0x300save SS1
```

```
# make sub system SS2new design SS2add_instance -name I0 -master
C1add_interface -name SS2B1 -interface_ref D_I -address_range 0x600 -type
SLAVEadd_interface -name SS2B2 -interface_ref D_I -address_range 0x400
-type MASTERadd_memory_map -name OM2add_requestor_map -memory_map OM2
-requestor_intf SS2B1add_target_map -memory_map OM2 -target_inst I0
-target_intf C1P1 -start 0x100
#target mapping with given range feedthruadd_target_map -memory_map OM2
-target_intf SS2B2 -start 0x200 -end 0x3ff -feedthru 0x100save SS2
# make design SS0new design SS0add_instance -name SS1_1 -master
SS1add_instance -name SS2_1 -master SS2add_interface -name SBIA
-interface_ref D_I  -address_range 0x600 -type SLAVEadd_interface -name
SBIB -interface_ref D_I  -address_range 0x600 -type SLAVEadd_address_path
-requestor_intf SBIB -target_instSS2_1 -target_intf SS2B1add_memory_map
-name OM0Aadd_requestor_map -memory_map OM0A -requestor_intf
SBIAadd_target_map -memory_map OM0A -target_inst SS1_1 -target_intf SS1A
-start 0x200add_memory_map -name OM0Badd_requestor_map -memory_map OM0B
-requestor_instSS2_1 -requestor_intf SS2B2add_target_map -memory_map OM0B
-target_inst SS1_1 -target_intf SS1B -start 0x00save SS0
# elaborate hierarchical memory map and generate flat# level memory map
view.gen_memory_map
```

The graphical representation of the above design-level memory map is shown in the following figure:

Figure 14-7 Design-Level Memory Map For the Above Example



Here, address of leaf instances are calculated in the following manner:

- From SBIA entry point

```
[0x200-0x2FF]: SS1/I0[0x300-0x3FF]: SS1/I1[0x400-0x4FF]: SS1/
I2[0x500-0x5FF]: SS1/I3
```

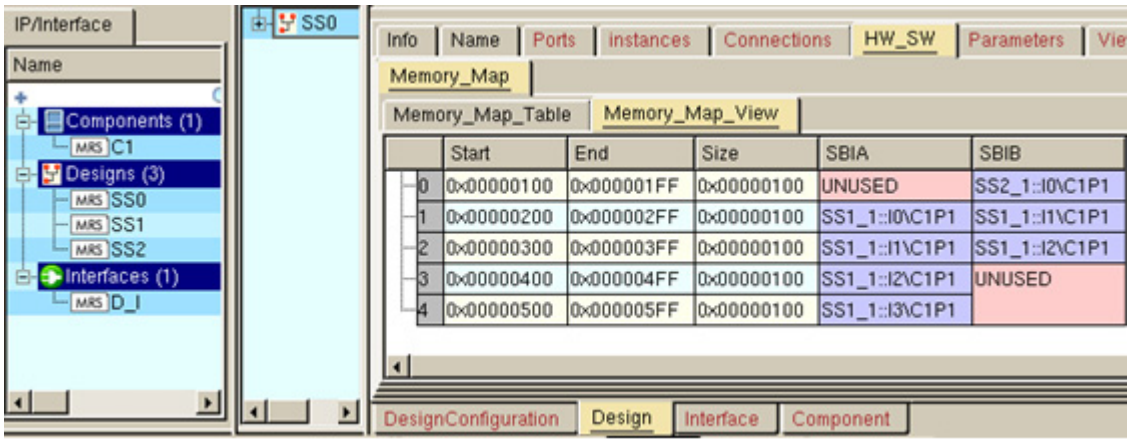
- From SBIB entry point:

```
[0x000-0x0FF]: VOID[0x100-0x1FF]: SS2/I0[0x200-0x2FF]: SS1/
I1[0x300-0x3FF]: SS1/I2[0x400-0x4FF]: VOID[0x500-0x5FF]: VOID
```

- Instances SS1/I0 and SS1/I3 are invisible (visibility is limited to 0x3FF).

The elaborated design-level memory map view (with feedthrough) for the above memory map is shown in the following figure:

Figure 14-8 An Elaborated Design-Level Memory Map View For Previous Memory Map



GenSys provides you a simpler way to add memory map data for a design, and simultaneously view the flat memory map view in a much more interactive way. You can do this in the Register/Memory-Map Builder dialog. For more details on this dialog, refer to the Viewing Graphical Representation of Register and Memory-Maps section.

Viewing Graphical Representation of Register and Memory-Maps

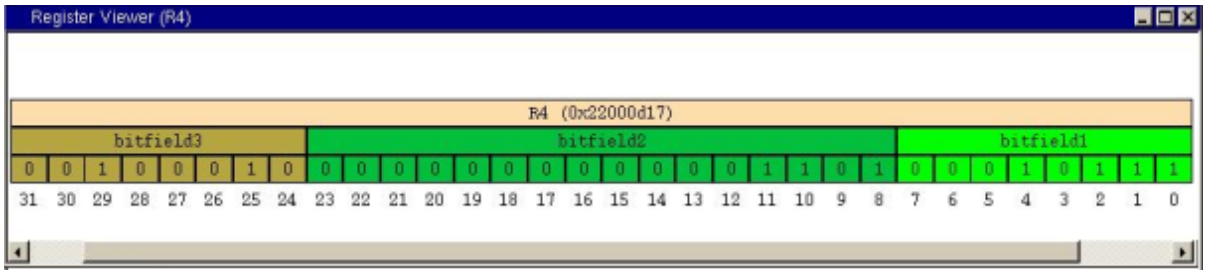
This section covers the following topics:

- Viewing Register Information
- Viewing Memory-Map Information

Viewing Register Information

When you open the editor dialog for a register, GenSys simultaneously displays the Register Viewer window of that register. The Register Viewer window provides a graphical representation for each register in which you can view the placements of various bitfileds added in that register. A sample Register Viewer window is shown in the following figure:

Figure 14-9 Register Viewer Window



The above dialog displays the graphical view of a register, R4. It also displays the reset value, 0X22000d17, for this register adjacent to the register name. In addition, this dialog displays the bitfields along with their reset values. Each access type in this dialog is highlighted in a different color.

Viewing Memory-Map Information

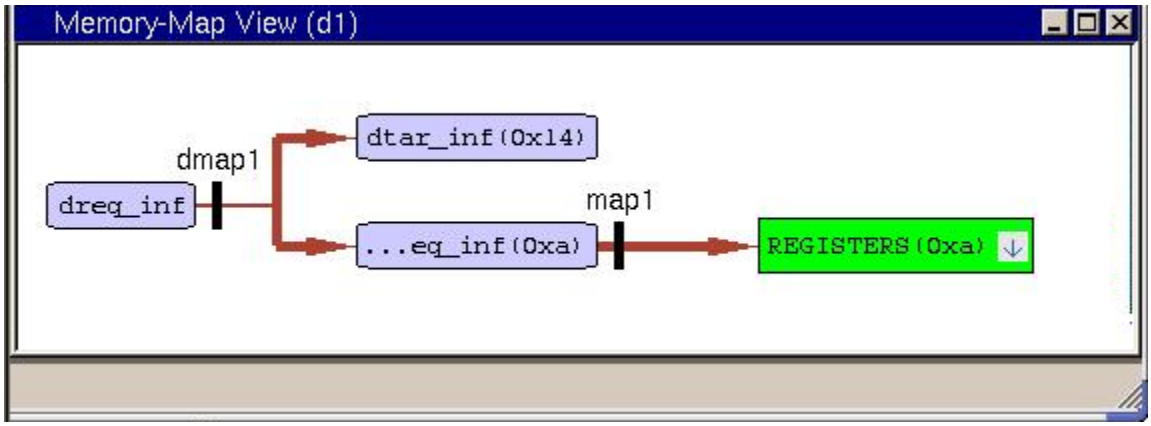
You can view the graphical representation of a memory-map in the Memory-Map View dialog. To view the memory-map, perform ANY of the following steps:

- Click the Memory map Viewer button (

-).
- Select Registers/Memory Map->Memory map Viewer option from the Tools menu.

A sample Memory-Map View dialog is shown in the following figure:

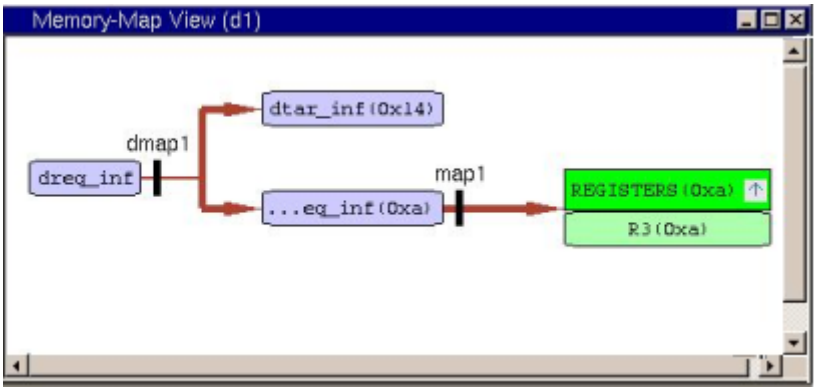
Figure 14-10 Memory-Map View Dialog



The above figure displays a detailed memory map view for the design, d1. Here, the memory map view contains two memory maps, dmapp1 and map1. All the interfaces are highlighted in blue color. In addition to the interfaces, the memory map view also displays the target registers (highlighted in green) of requestor interface. To view all the registers, click the

button adjacent to the REGISTERS box. This displays all the target registers of the particular requestor, as shown in the following figure:

Figure 14-11 Target Registers of a Particular Requestor



Here, only one register, R3, is present for the requestor. To edit the information of this register, you can double-click on this register directly from the memory-map viewer dialog. This displays the Register Editor dialog where you can make the required modifications.

You can increase or reduce the magnification of the memory-map view by dragging the mouse (with the left button pressed) in appropriate directions. This allows you to zoom-in and zoom-out the memory-map view accordingly depending upon the length of mouse drag. You can also fit the entire contents in the available window size by selecting the Zoom Fit option while dragging the mouse in an appropriate direction.

A

The Group/Ungroup Feature

The Group/Ungroup feature in GenSys is a powerful tool for hierarchy manipulation. However, it necessarily involves complex algorithms that might make the results appear non-intuitive in a small number of cases.

Functions and Characteristics of Group/Ungroup Feature

This section covers the following topics:

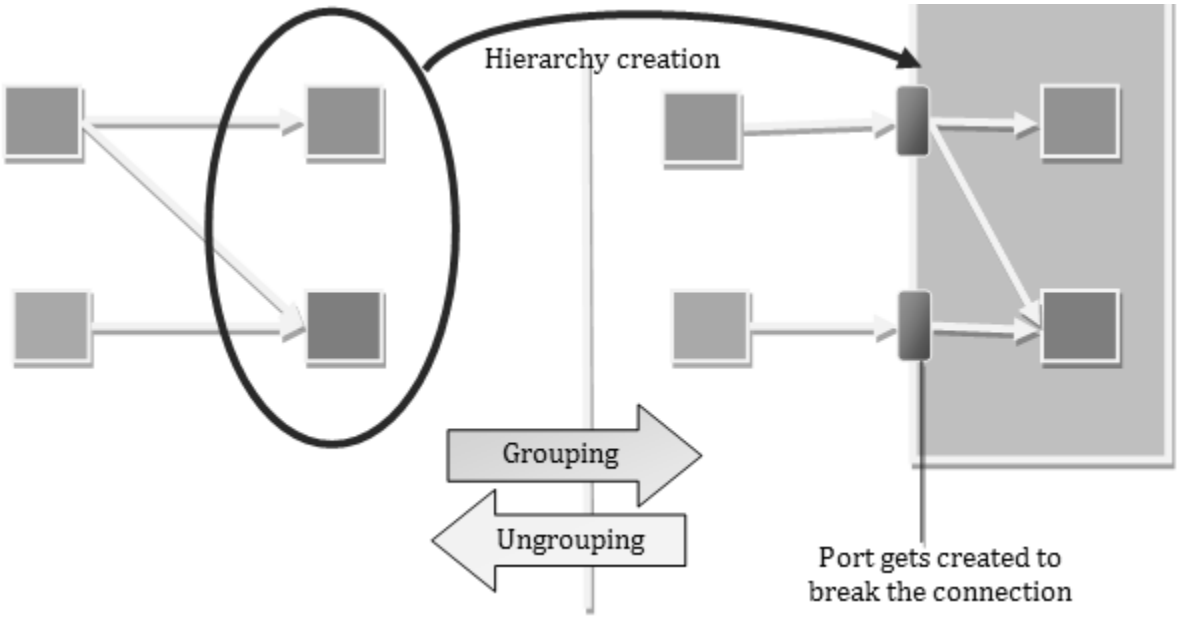
- Identifying the Intended Use Areas
- Use Model
- Full Logic Equivalence
- Effect on Connections Across the Subsystem Hierarchy

Identifying the Intended Use Areas

GenSys provides the Group/Ungroup feature for flexible hierarchy creation and dissolution. In any SoC flow, the design creation is distributed across various teams. Often there are situations in which a part of the design needs to be cordoned off and handed over to another design team. In such cases in which the design started off as flat, a hierarchy needs to be created and handed off to the new team. The Group/Ungroup feature facilitates this so that another team can work on the internals of the hierarchy, and give back the modified design to the original developer, who may then break the boundaries to assimilate it back into the main design.

Use Model

Grouping refers to the consolidation of multiple instances into a hierarchy. Connections are redrawn to break at the boundary of the new block. This is illustrated in the following figure:



Full Logic Equivalence

The Group/Ungroup feature is designed for full logic equivalence. This means that each stage of grouping/ungrouping, if exported to HDL, will produce a design that is totally equivalent w.r.t. the connectivity. Therefore, in this appendix when connections being deleted, or propagation of tieoffs/opens, the references are only with respect to the GenSys representation of the design, and have no impact on the equivalence of the generated RTL at each stage. This allows you to go through unlimited group and ungroup transformations. Of course, the hierarchies created through the grouping operation shows up as intermediate hierarchies in the generated RTL. There may be some variations in the HDL net names generated during successive grouping/ungrouping operations. Other than that, the HDL is logically equivalent at every stage.

Effect on Connections Across the Subsystem Hierarchy

When a connection spans across a subsystem boundary in a grouping operation, GenSys breaks the connection to create identical connections on both the sides of the boundary connected to a new port at the boundary. Ungrouping such hierarchies generally produce designs that are identical to what existed before grouping. However, in case of hierarchies not created through grouping, it is common to find an intersection of dissimilar connections at the subsystem boundary. In particular, the meeting of logical and adhoc connections at the subsystem boundary is one such situation where GenSys needs to take decisions, trying

to preserve the logical connections. In many cases, GenSys proceeds by performing a local elaboration, therefore reducing the logical connections to the corresponding mapped adhoc connections. This step creates new connections when ungrouping and dissolve the old ones. However, it is important to note that this does not break the logical equivalence in any way. Interfaces and logical connections are GenSys artifacts that are anyway reduced to their mapped equivalents during the actual step of elaboration and HDL export.

Principles Governing the Grouping/Ungrouping Feature

This section covers the following topics:

- Principles Governing Grouping
- Principles Governing Ungrouping

Principles Governing Grouping

The grouping flow broadly consists of the following steps:

1. A new subsystem is created, instantiating the selected design objects inside it. If the RTL is generated, this hierarchy shows itself in the generated hardware.
2. The inbound and outbound connections are broken at the boundary of the subsystem, and new connections are appropriately created.
3. Tieoffs and Opens are optionally propagated to the subsystem boundary.

If a pin of an IP has a parameter associated with it, then during grouping, GenSys evaluates the parameter value of the port created on the group boundary.

If you do not want GenSys to evaluate the parameter value for such ports during grouping, set the value of the -eval_param_on_group argument of the set_gensys_options Tcl command to N. In this case, GenSys retains parameter expressions in MSB or LSB of the port created on group boundary and also adds the parameter in the group boundary.

Principles Governing Ungrouping

The ungrouping action on a subsystem dissolves the hierarchy, and places the design objects inside at the same level as the one at which the subsystem originally was.

Since there are new objects (connections, tieoffs, etc.) created as a result of the ungrouping action, there are often multiple ways to ungroup. Decisions are taken by the tool based on certain principles discussed below.

- Ungrouping a subsystem should follow connectivity.

The terminals of the {inside - subsystem - outside} net should be connected as {inside - outside}, then {inside - subsystem}, and {subsystem - outside} connections should be deleted. When all the connections have been bypassed and removed, the subsystem should be deleted.

- Ungrouping a subsystem implies that any connectivity terminating as open on a subsystem is no longer required.
- Ungrouping a subsystem implies any active connectivity terminating on a subsystem (for example, an active tieoff) should be propagated to an appropriate net terminal. Unused connectivity priority information (for example, currently unused tieoffs, defaults) on the subsystem is lost.
- Ungrouping a subsystem implies that any information on the subsystem (subsystem) is no longer require. For example, properties or unused connectivity priority information (for example, unused tieoffs, defaults).
- Ungroup behavior other than the above discussed points is currently not supported.

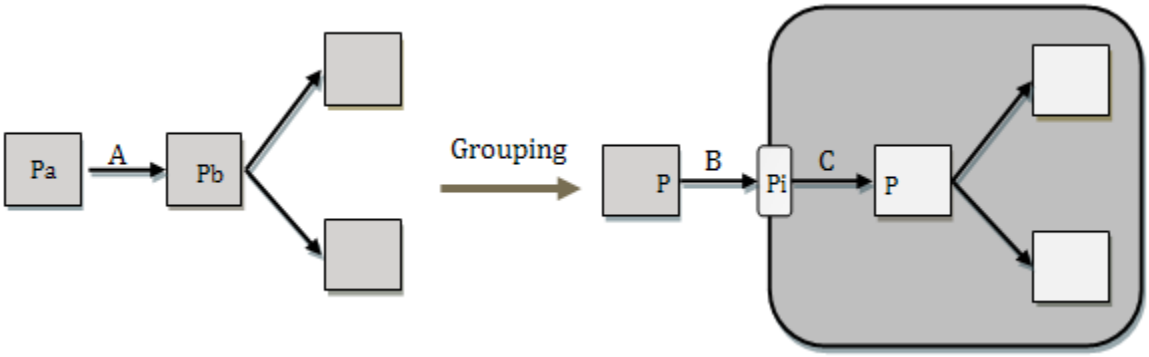
Scenarios of Grouping

This section discusses various scenarios of grouping. It covers the following topics:

- Single Adhoc/Interface Connections Scenario
- External Fanout Scenario
- Internal Fanout Scenario

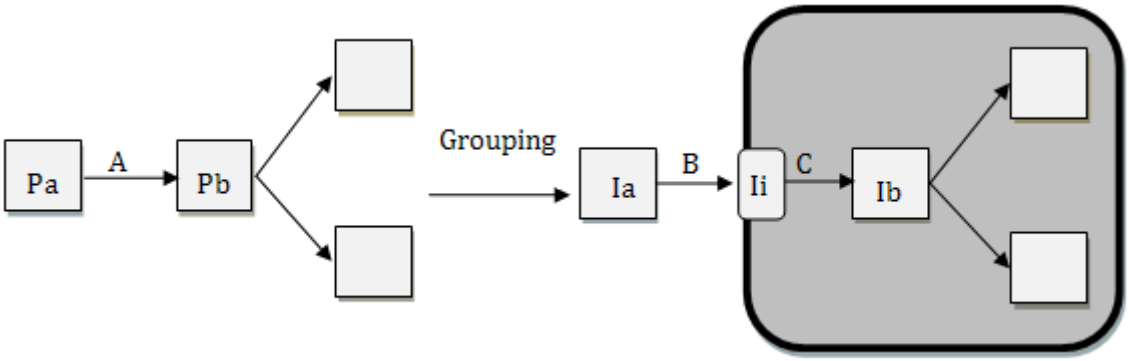
Single Adhoc/Interface Connections Scenario

Single connections, that is, connections without fanout, simply get broken at the boundary by the creation of a new port, as shown in the following figure:



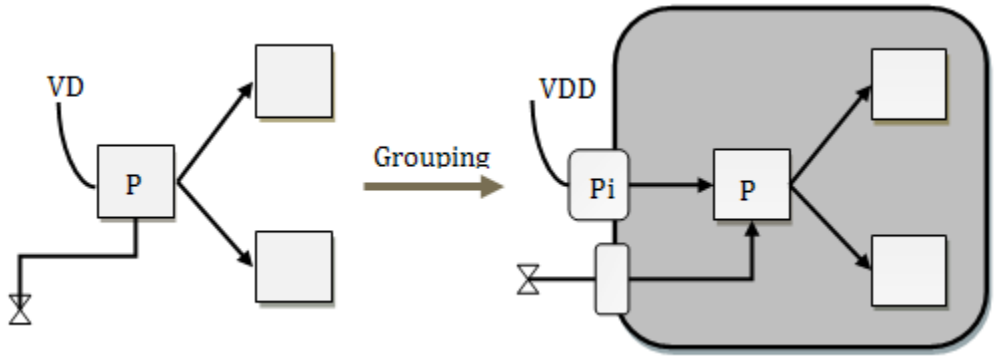
In the above case, the adhoc connection, scalar or vector, breaks at the boundary, a new port gets created, and the connections made on the either side of the port. Attributes of connection, A, in the above figure are copied to both the connections, B and C.

However, in case of an interface connection, an interface is created on the boundary instead, and logical connections are made on both the sides of the newly created interface, as shown in the following figure:



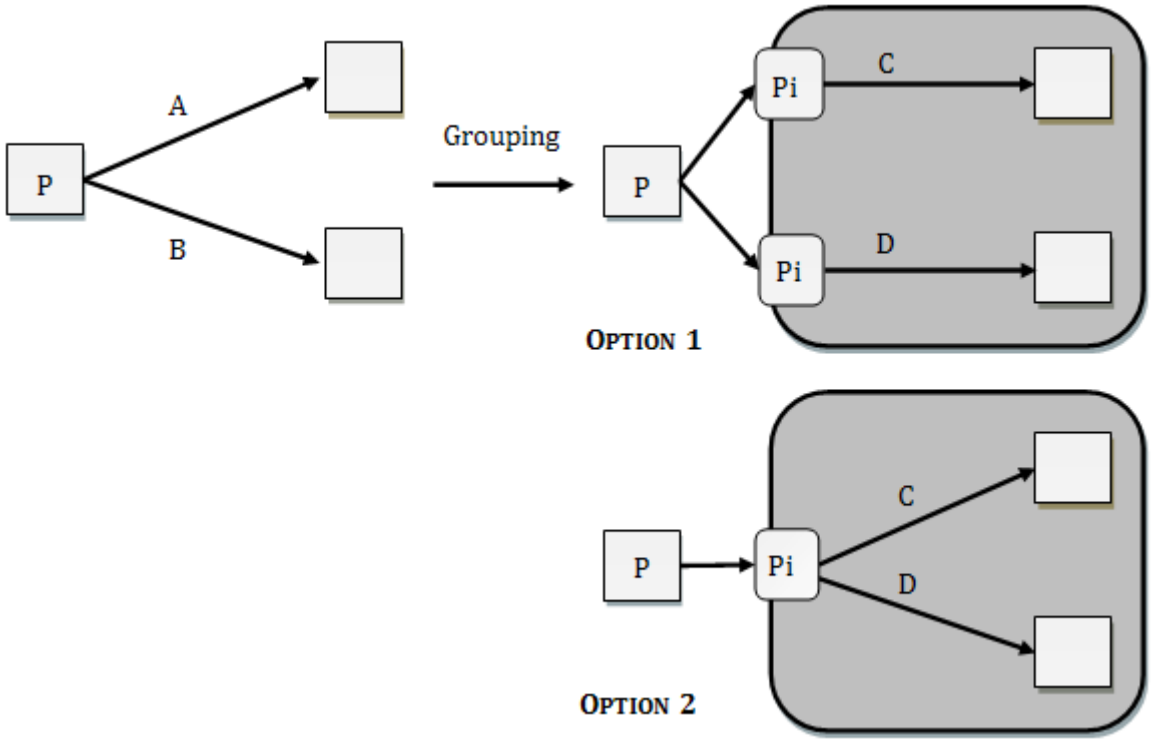
Tieoff and Open Connections

During grouping, tieoffs and opens have the option of being propagated to the subsystem boundary, or remain attached to the port within the hierarchy. The following figure shows the case in which the tieoff and open is propagated to the boundary:



External Fanout Scenario

Consider a scenario illustrated in the following figure:

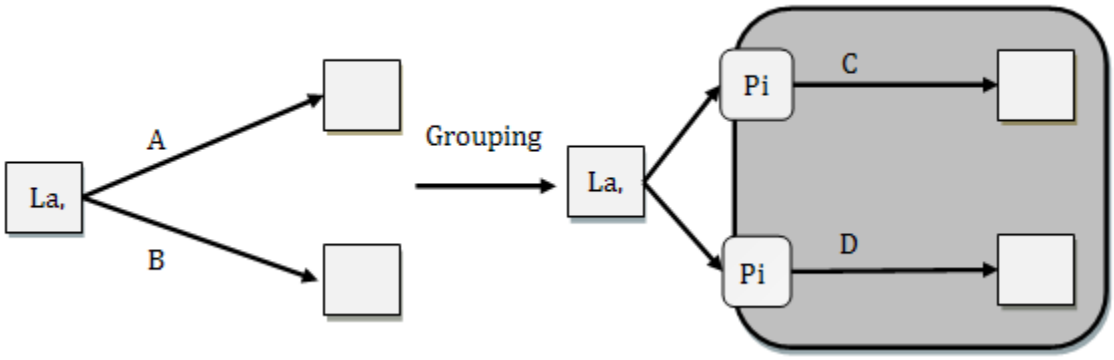


As shown in the above figure, grouping with external fanout of adhoc connections can produce two different hierarchies. This is under user control with the help of the inbound/ outbound option provided in the Grouping dialog. For more details on this dialog, refer to *GenSys User Guide*.

In option 2 in the above figure, the attributes of A and B are transferred into C and D, respectively. The external connection between the ports, Pa and Pi, does not carry any attributes. During ungrouping, these two connections merge back into a single connection, and the attributes of the new connection are a union of the attributes on both sides. That way, the attributes remain unchanged, if no changes were made to them post grouping.

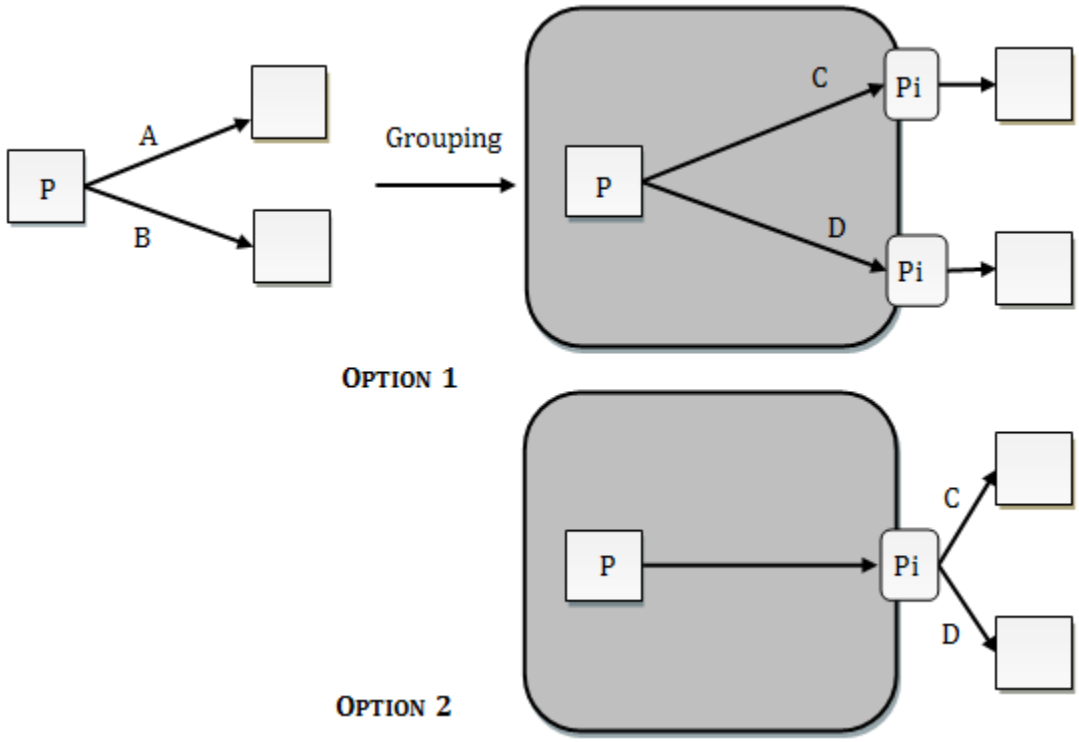
If the above case involved a logical connection (full or spliced) and interfaces instead of adhoc connections, then only Option 1 is followed. That is, in case of interface connections, GenSys does not collapse the two connections into a single connection from the source with internal fanout.

In addition, in case there is a logical connection from an interface and an adhoc connection from one of the mapped ports into the hierarchy, there is no collapsing either. This is shown in the following figure:



Internal Fanout Scenario

Consider a scenario illustrated in the following figure:



As in the case of external fanout of adhoc connections, there are similar options in the internal fanout scenario. The figure above shows the two options. In option 2, the attributes of A and B get transferred to C and D, respectively, whereas the connection inside does not have any attribute values.

In a combination of external and internal fanout situations, user can further guide the port creation process further by choosing the option of minimum or maximum ports creation.

In case of an internal fanout of an interface connection, the fanout is propagated to an interface on the boundary, similar to the Option 2 of the adhoc case shown in the above figure. This also happens even if one of the connections is a spliced connection.

If there is an outbound logical connection and an adhoc connection from a corresponding mapped port, the connections are broken independently at the subsystem boundary to create a port and an interface.

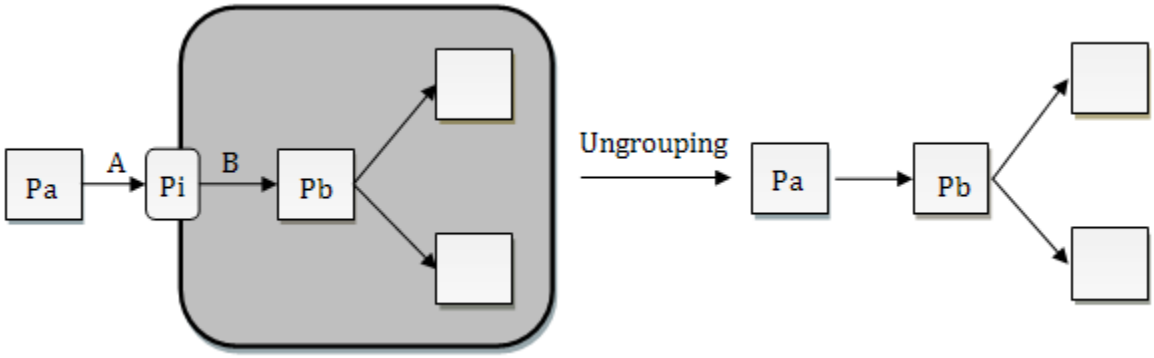
Scenarios of Ungrouping

This section discusses various scenarios of ungrouping. It covers the following topics:

- Ungrouping Scenario Within Adhoc and Outside Adhoc
- Ungroup Scenario in Case of Adhoc and Interface/Spliced
- Ungroup Scenario in Case of Interface/spliced and Interface
- Ungrouping Scenario in Case of Tieoffs
- Ungroup Scenario in Case of Open
- Ungroup Scenario in Case of Nets
- Ungroup Scenario in Case of Feedthroughs and Feedbacks

Ungrouping Scenario Within Adhoc and Outside Adhoc

Consider the scenario illustrated in the following figure:



This is the simplest scenario in which there is an adhoc port at the subsystem boundary, with adhoc connections proceeding to the block instance within as well as without. The port dissolves to make a direct adhoc connection between inside and outside block instances.

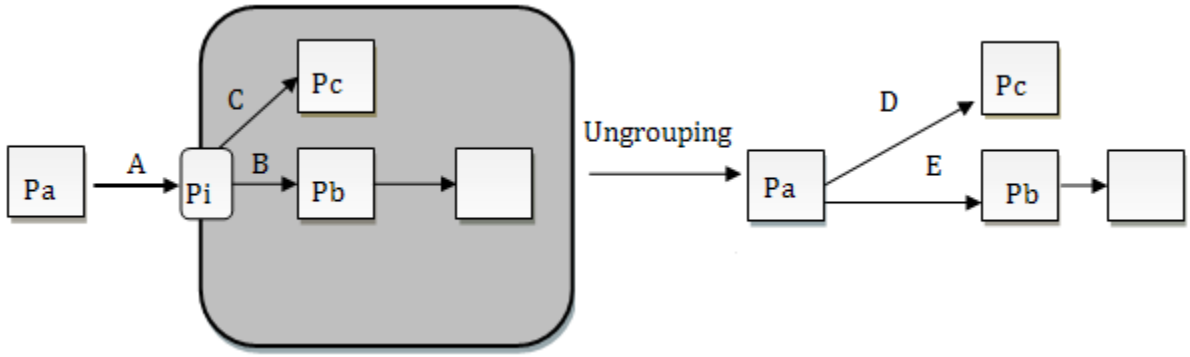
Attributes are copied from one of the connections to the final connection. The following table defines what gets copied:

Direction of Pa	Direction of Pb	Source of attributes
OUT/INOUT	IN	Connection A
IN	OUT/INOUT	Connection B

Direction of Pa	Direction of Pb	Source of attributes
INOUT	INOUT	Connection A (the one outside)

A More Complex Case of Adhoc-Adhoc

It may happen that multiple sizes of adhoc connections converge at the subsystem boundary port. For example, consider the scenario illustrated in the following figure:



Consider the sizes of Pa and Pi as [7:0]. The connection in the hierarchy is shown in the following table:

Connection	From	To
A	Pa[7:4]	Pi[7:4]
B	Pi[5:2]	Pc[3:0]
C	Pi[4:3]	Pb[1:0]

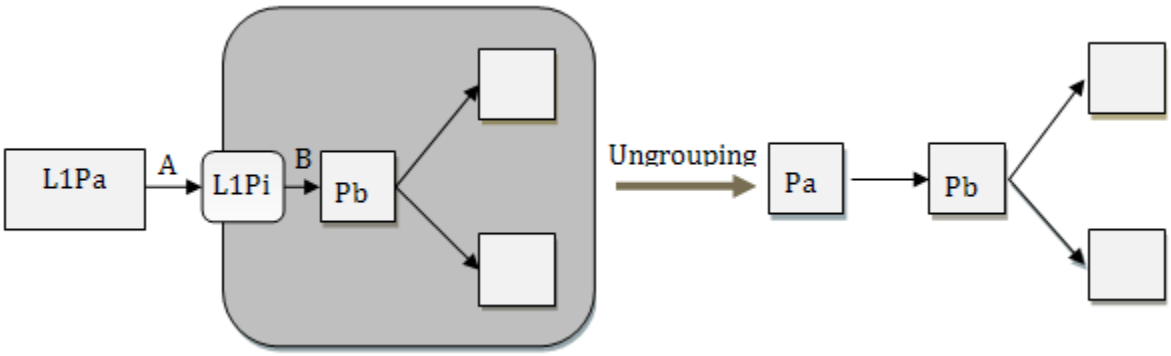
Then, the ungrouping will produce the following new connections

Connection	From	To
D	Pa[5:4]	Pc[3:2]
E	Pa[4]	Pb[1]

In this case, the attributes of connection A are copied to D and E.

Ungroup Scenario in Case of Adhoc and Interface/Spliced

Consider the scenario illustrated in the following figure:



The consolidation of the logical connection and an adhoc connection is performed by reducing the logical to a physical connection. Therefore, in the above example, lets say, connection B is between the actual ports, Pi and Pb, within the subsystem. Also, the logical port, L1, of the interface is mapped to Pa on one side of the connection, A, and Pi to the other. In that case, the ungrouping will cause the logical connection to disappear, and the adhoc connection alone will remain.

Attributes are copied from one of the connections to the other, based on the following table:

Direction of Pb	Source of Attributes
OUT/INOUT	Connection B
IN	Connection A

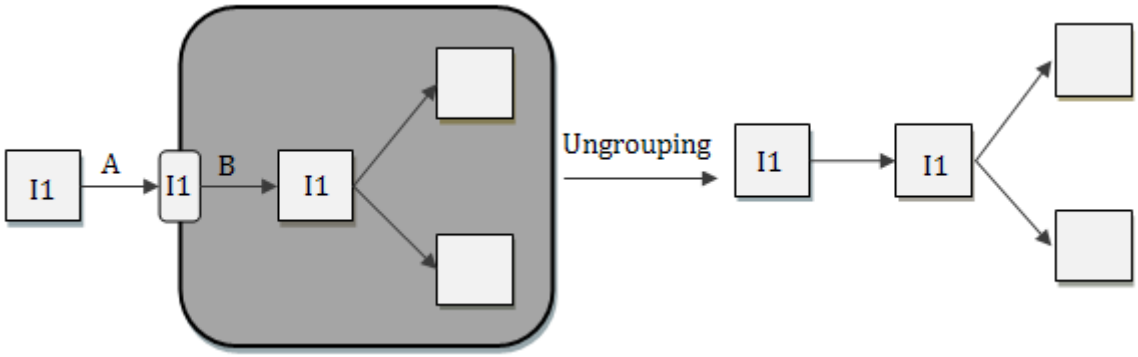
Where attributes are copied from a logical interface connection to an adhoc connection, it is presumed that only the relevant attributes are copied.

The spliced case is not very different from the above, and the actual mapped pins in the splice are connected to the matching adhoc connections.

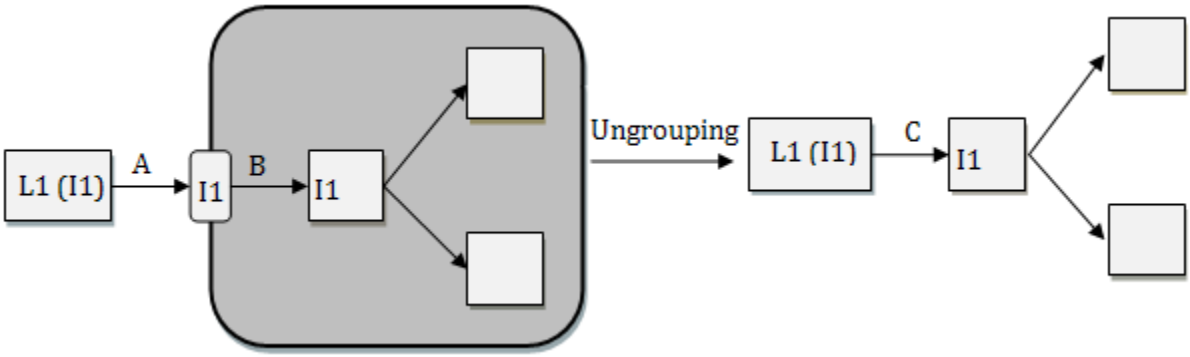
Ungroup Scenario in Case of Interface/spliced and Interface

In case of an interface to interface connection, the interface at the boundary is dissolved to make a direct interface-to-interface connection.

Consider a scenario, as shown in the following figure:



If either of the connections is a splice, the resultant connection is a spliced connection, as shown in the following figure:



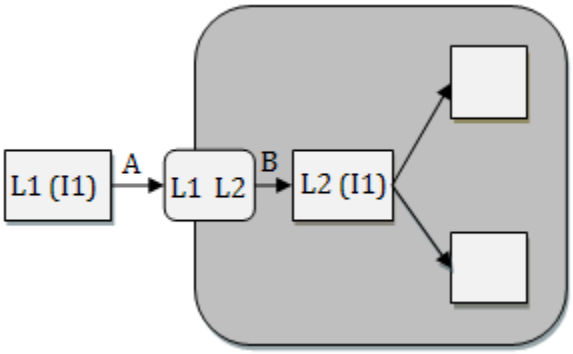
In the above figure, A is a spliced connection, whereas B is a straight interface-to-interface connection. After ungrouping, C is a spliced connection like A.

For attributes on the final connection, we do not have the same distinctions available as in the adhoc cases, where port directions determined the driver side. Here, we use a convention that the connection emanating from an interface with multiple connections (whether within or without the subsystem) will dictate the attributes of the final connection. However, if neither of the interface ports is driving multiple nets, we consider the external (outside the subsystem) as the source of the attributes.

In the case of an interface + splice, the net connected to the interface always determines the attributes to be copied to the final spliced connection.

Special case of Ungrouping Spliced and Spliced

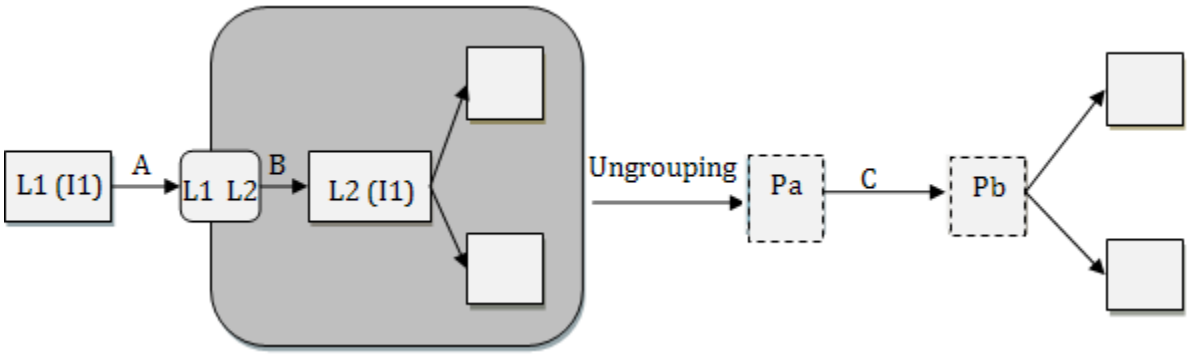
Consider the scenario illustrated in the following figure:



Let us assume that connection, A, connects a splice containing the logical port, L1. Connection, B, within the subsystem connects splices with the logical port, L2. Now, when we ungroup, it needs to be determined which spliced connection to propagate all the way between the ports of the block instances. It appears that ungrouping would cause the connections to simply disappear, since there is no common logical port between the two splices.

However, L1 and L2 may be mapped to the same physical port on the boundary. If that happens, there is a physical adhoc connection replacing two logical connections.

Consider the following figure:



This shows the complexity that comes in because of the intersection of logical and physical domain spaces. The only way to resolve such cases is to reduce the logical to actual, which is what GenSys does.

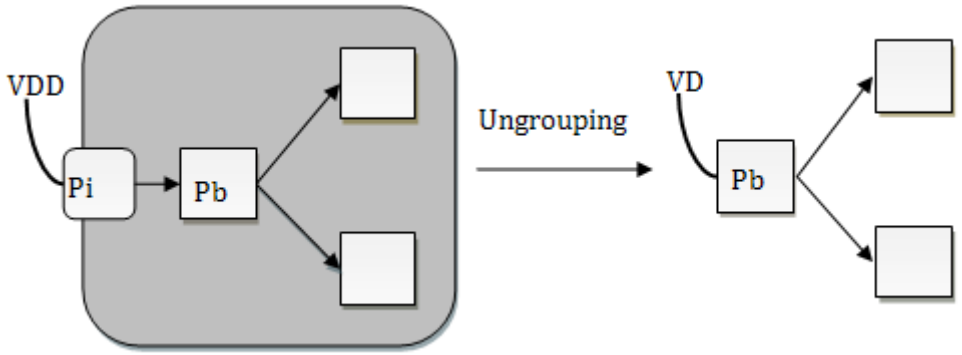
Ungrouping Scenario in Case of Tieoffs

This section covers the following scenarios:

- Ungroup Scenario in Case of Adhoc scalar and Tieoff
- Ungroup Scenario in Case of Adhoc Vector and Tieoff
- Ungroup Scenario in Case of Interface/Spliced and Tieoff
- Treatment to Attributes in Tieoff
- Temporary Tieoffs
- Ungroup Scenario in Case of Nets
- Ungroup Scenario in Case of Feedthroughs and Feedbacks

Ungroup Scenario in Case of Adhoc scalar and Tieoff

Consider the scenario illustrated in the following figure:



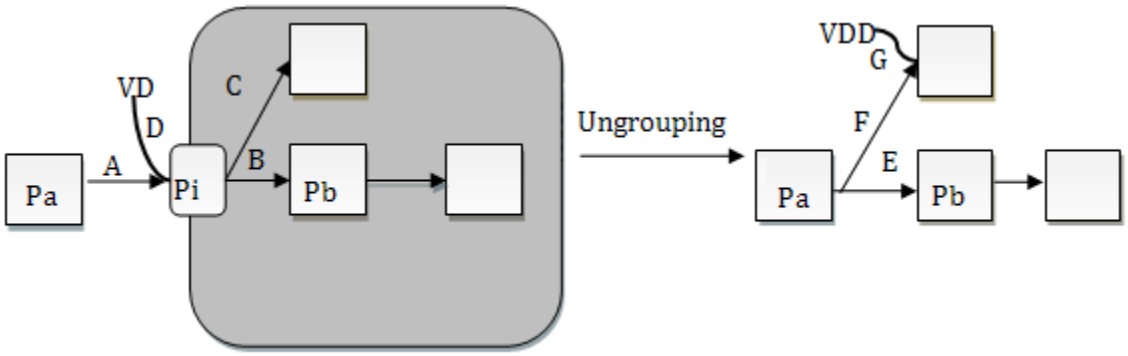
In the case of a tieoff, whether on an input port from outside, or an output design port within the subsystem, the tieoff gets propagated to the connections affected.

If there was a fanout at Pi, the tieoff would propagate to each of the connections.

Ungroup Scenario in Case of Adhoc Vector and Tieoff

In case of a vector port, only those specific bits that are tied off are affected. The tieoff propagates irrespective of other connections on the port, which are affected as per their own rules.

Consider the scenario illustrated in the following figure:



In the above example, Pa and Pi are vector ports. The connections are as follows:

Connection	From	To
A	Pa[7:4]	Pi[7:4]
B	Pi[7:5]	Pb[3:1]
C	Pi[4:3]	Pc[1:0]
D	Pi[3]	Tiedoff to VDD

After ungrouping, the new connections are as follows::

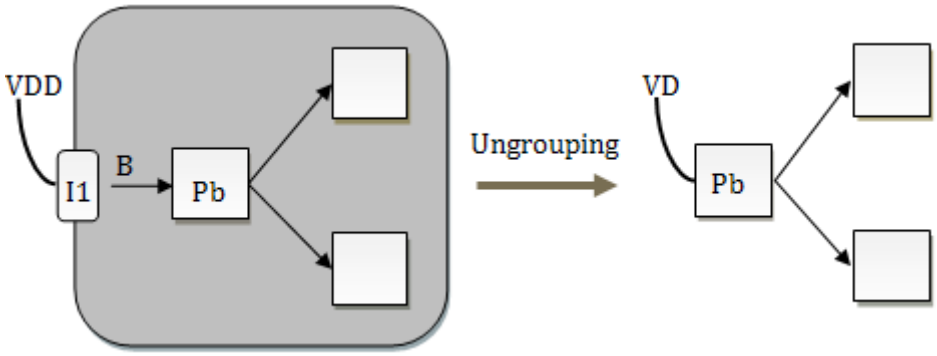
Connection	From	To
E	Pa[7:5]	Pb[3:1]
F	Pa[4]	Pc[1]
G	Pc[0]	Tied off to VDD

This is equivalent to first applying a tieoff and propagating it to the destination. Next, we connect A to the port, and use adhoc-adhoc rules to ungroup.

Ungroup Scenario in Case of Interface/Spliced and Tieoff

The tieoff is propagated into the subsystem along the spliced connection. The spliced connection itself is deleted if there is nothing else connected to the boundary port, or else gets connected to the inbound connection.

Consider the scenario illustrated in the following figure:



If an adhoc connection joins the interface from outside with a splice connection inside, besides a spliced tieoff at the interface, the situation can be considered a combination of the above example and section, Ungroup Scenario in Case of Adhoc and Interface/Spliced, except that the splice is inside. Thus, the splice is reduced to the physical ports, and the treatment is similar to section, Ungroup Scenario in Case of Adhoc scalar and Tieoff.

Treatment to Attributes in Tieoff

Whenever the tieoff is propagated through an existing adhoc/spliced/interface connection within the subsystem, the attributes from that specific connection are copied into the tieoff connection.

Temporary Tieoffs

The treatment to temporary tieoffs is similar to tieoffs except that it is a temporary tieoff that gets propagated.

Ungroup Scenario in Case of Open

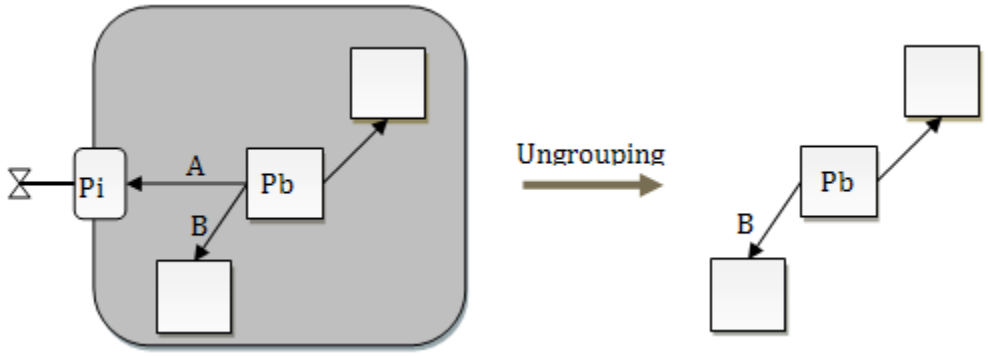
An open at an output port is not propagated to the other end of connections, as this can affect the other connections on those ports. Therefore, the only impact of open is that the internal connection is dissolved.

This section covers the following scenarios:

- Ungroup Scenario in Case of Adhoc scalar and Open
- Ungroup Scenario in Case of Adhoc vector and Open
- Ungroup Scenario in Case of Interface/Spliced and Open
- Treatment to Attributes in Open
- Temporary Open

Ungroup Scenario in Case of Adhoc scalar and Open

Consider the scenario illustrated in the following figure:



In this case, the open causes connection, A, to be deleted. Note that unlike a tieoff, an open cannot be propagated to the other end. In the case above, if the open were to be propagated to the other end of connection, A, the precedence rules in GenSys cause connection, B, to be deleted during elaboration, since there would be an open connection at Pb.

Ungroup Scenario in Case of Adhoc vector and Open

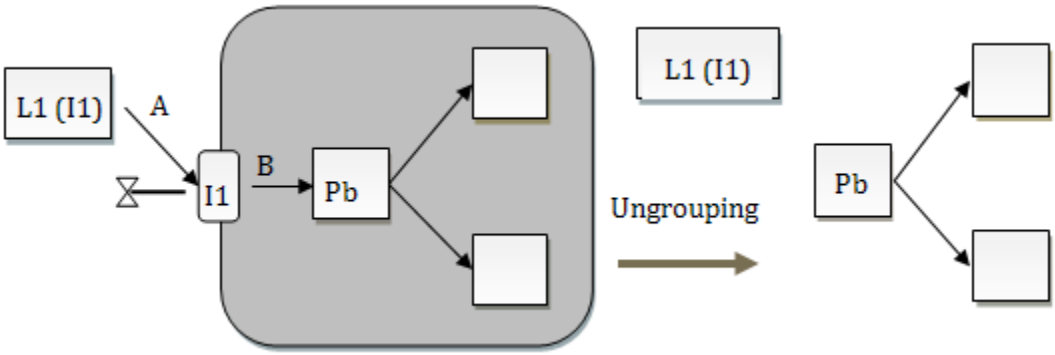
In such cases, connections to the specific bits of the vector, which are open, are deleted just as in the scalar case.

Ungroup Scenario in Case of Interface/Spliced and Open

If an interface at the subsystem boundary has an open connection on it, the connections to the interface are deleted as in the adhoc case.

If an open resides on a splice, then as in the adhoc vector case, connections to the splice are deleted. However, connections to the rest of the interface are dealt with respectively.

Consider the scenario illustrated in the following figure:



Treatment to Attributes in Open

All connections to a port/interface that is open are lost and so are the attributes of those specific connections.

Temporary Open

A temporary open is treated in the same way as a normal open.

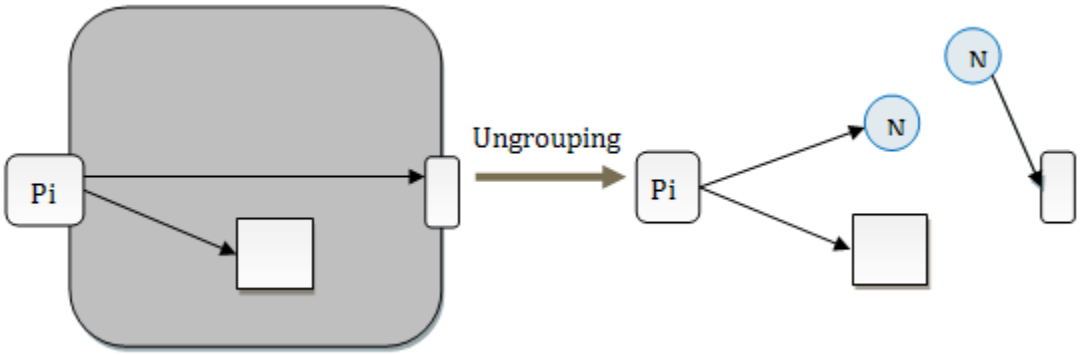
Ungroup Scenario in Case of Nets

When adhoc ports are connected to labeled nets, the ungroup operation does not perform any reconnections. The labels and the nets are preserved as they are, for the connection at elaboration time.

Ungroup Scenario in Case of Feedthroughs and Feedbacks

Feedthroughs and feedbacks are broken, and each end is connected to a logical net.

Consider the scenario illustrated in the following figure:



Parameters and Instance Names During Ungrouping

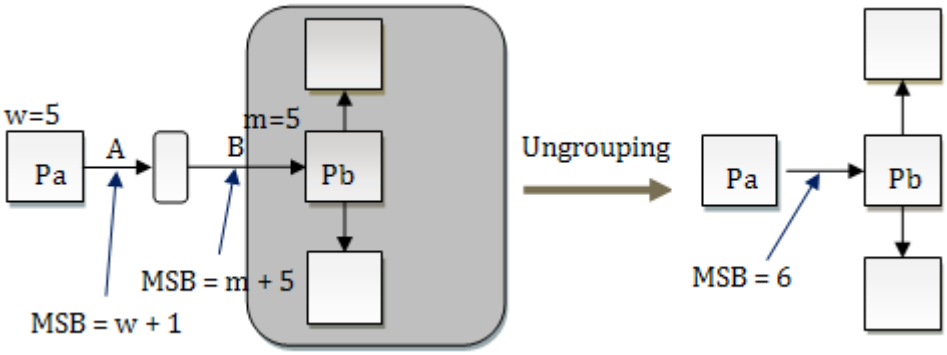
This section covers the following topics:

- Treating Parameters During Ungrouping
- Treating Instance Names During Ungrouping

Treating Parameters During Ungrouping

Parameters, like connection width and alignment, are used during ungrouping to create the new connections. Since parameters can be dependent on other component parameters, their value needs to be evaluated before new connections can be created.

Consider an example, as shown in the following figure:



In the above figure, A and B are parametrized connections. During ungrouping, the parameters are evaluated, and A of size [6:0] is connected to B of size [10:0]. Assuming LSB based alignment, the final connection is a vector that connects Pa[6:0] to Pb[6:0]. In this case, Pb[10:7] remain open.

Also, the final connection is not parametrized. The MSB carries the evaluated value of 6.

The attributes on the final connection are copied from A, including align, deltaDelay, description, etc. "align", though used during the above evaluation process, is also copied onto the connection.

Treating Instance Names During Ungrouping

If an instance in the subsystem hierarchy has the same name as one of the instance in the level above, there will be a name conflict when the hierarchy is dissolved, and the internal objects are brought into the level above. GenSys issues a warning, and changes the name of the instance inside the hierarchy to a unique name before bringing it at par with the objects above.

B

Using Configurable IPs

To make an IP reusable, it should be easily adaptable for use in different architectures. To meet this requirement, an IP should be easily configurable to suit different specifications. For example, the requirement of an SoC architecture may be that data and address ports of an IP should be 32 bits. For another Soc architecture, this requirement may change to only 16 bits. To accommodate such specifications, IPs should be easily configurable.

Configurable IP Definition

A configurable component or IP has some parameters that can be configured by the design engineer, and these parameters can be further configured in the RTL design.

Simple Configuration

You can configure an IP by through the parameters of that IP. For information on how to use parameters, see the Parameters tab section in Working with Parameters or see the add_parameter Tcl command in GenSys Tcl Commands Reference.

Simple configurable IP can be created by using simple parameters. For example, the MSB of the ports can be dependent on parameter P1 of a component.

HDL Parameters

Component/IP hardware parameters allow you to configure a component.

Only *int* and *enum* type parameters of GenSys can be mapped to HDL parameters in that case you have to set the rtl field of the parameter to Y (YES).

Complex Configuration

You can use parameters to make a design configurable. For more information on how to use parameters, see Capturing Component Register Information Using GUI or see the *add_parameter* Tcl command in GenSys Tcl Commands Reference.

Highly configurable IP can be created by using complex parameters.

Using Configurable IP

The configurable IP or components can be instantiated inside the design and desired parameters values for that IP can be given.

This can be done by the following method:

- Using the Tcl command *set_parameter* (See GenSys Tcl Commands Reference for more details).

  These parameters can also be reset by using reset_parameter.

- Using GenSys GUI by right clicking on the component instance in the IP Browser and selecting Reconfigure Instance menu option.

Compiler Directive Support

Verilog provides support for compiler directives, such as `ifdef, `endif, and `ifndef to include optionally lines of a Verilog HDL source description during compilation. If the macro called out by the `ifdef command has been defined, that code is compiled. If not, the code following the command `else is compiled. If the `elsif directive exists, the compiler checks for the definition of the corresponding macro name. The `define construct defines the macros which impact the `ifdef command and is typically applicable globally.

GenSys provides mechanisms to recognize these Verilog pre-processor directives and package them as configurable IPs. From a packaging perspective, only the ports and the parameters can be conditionally present within `ifdef commands. Currently, GenSys does not support nested pre-processor directives.

Verilog netlisted from GenSys retains the compiler directive information.

GenSys stores a visibility element with each of the port and parameter that is imported from the RTL. The visibility element is empty by default. This implies that the port/parameter is not inside any conditional macro.

In case a port is inside an `ifdef or `ifndef macro in Verilog, it is imported inside GenSys and has the visibility flag set as ifdef::<macro name> or ifndef::<macro_name>, respectively.

User can also specify macro visibility on ports and parameters by using the -macro_visibility option of the add_port and add_parameter Tcl commands. For details about these Tcl commands, refer to the GenSys Tcl Commands Reference.

Since IP-XACT doesn't have a concept of visibility, Gensys will store this information as Atrenta Vendor Extensions.

During instantiating the IP, you have the option to define macros of the IP. All instances of the IP need to have the same set of macro defined else and ERROR message is thrown and originally defined set of macros will be retained for the IP. User can define macros of master IP during instantiation by editing the configurability field under the INFO tab of the GUI instance.

You can define macros of the master IP by using the -configurability option of the add_instance Tcl command. For details about this Tcl commands, refer to the GenSys Tcl Commands Reference.

Example

Consider the following RTL code, which has two macro's X and Y:

```
module comp (`ifndef Y
             p5,
           `else
```

```
            p6,
         `endif

         `ifdef X
            p1,
         `else
            p2,
         `endif

               p4);
`ifndef Y
inout   [2:0]   p5;
`else
inout   [2:0]   p6;
`endif

`ifdef X
inout   [2:0]   p1;
`else
inout   [2:0]   p2;
`endif

inout   [2:0]   p4;
endmodule
```

Assume, GenSys has packaged this RTL as component comp using load_rtl. To instantiate the above component in a design, enter the following Tcl commands:

```
new design des
add_port -name d1
add_instance -name inst -master comp
run GenerateVerilog
```

Here, instance of the IP has not defined any macro.

The Verilog file generated is:

Component

```
module  comp (`ifndef Y
            p5,
         `else
            p6,
         `endif

         `ifndef X
            p2,
         `else
            p1,
         `endif

            p4);
`ifndef Y
inout   [2:0]   p5;
```

```
`else
inout   [2:0]   p6;
`endif

`ifndef X
inout   [2:0]   p2;
`else
inout   [2:0]   p1;
`endif

inout   [2:0]   p4;
endmodule
```

Design

```
module des ();
wire    [2:0]   UNINTENTIONAL_OPEN_comp0_p5_0;
wire    [2:0]   UNINTENTIONAL_OPEN_comp0_p6_0;
wire    [2:0]   UNINTENTIONAL_OPEN_comp0_p2_0;
wire    [2:0]   UNINTENTIONAL_OPEN_comp0_p1_0;
wire    [2:0]   UNINTENTIONAL_OPEN_comp0_p4_0;

 comp    inst (
`ifndef Y
             .p5(UNINTENTIONAL_OPEN_comp0_p5_0),  //IO:3
`else
             .p6(UNINTENTIONAL_OPEN_comp0_p6_0),  //IO:3
`endif

`ifndef X
             .p2(UNINTENTIONAL_OPEN_comp0_p2_0),  //IO:3
`else
             .p1(UNINTENTIONAL_OPEN_comp0_p1_0),  //IO:3
`endif

             .p4(UNINTENTIONAL_OPEN_comp0_p4_0)   //IO:3
                 );
endmodule
```

Now, assume you have defined a macro X for comp by either changing the configurability column under the Info tab for instance inst. Alternatively, you can specify -configurability {X}with the add_instance Tcl command for instance inst.

Then generated RTL is:

For component:

```
`define X
module  comp (`ifndef Y
             p5,
          `else
```

```
            p6,
         `endif

         `ifndef X
            p2,
………….
endmodule
```

For the design, the RTL is:

```
module des ();
……————..
……————..
 comp    inst (
            .p5(UNINTENTIONAL_OPEN_comp0_p5_0),  //IO:3
            .p1(UNINTENTIONAL_OPEN_comp0_p1_0),  //IO:3
            .p4(UNINTENTIONAL_OPEN_comp0_p4_0)   //IO:3
                  );

endmodule
```

Therefore, here the difference over the previous scenarios is:

- In master component comp, an additional `define is generated.
- In port map of instance, only the ports that are visible after macro X is defined, have been mapped.